A Voting Median Base Algorithm for Approximate Performance Monitoring of Wireless Sensor Networks

By
Yaqoob J. AL Raisi

A doctoral thesis submitted in partial fulfillment of the requirements for the award of Doctor of Philosophy of Loughborough University

February 2008

Dedicated to

My parents and my family

Acknowledgment

I would like to take this opportunity to thank the help and support I have received over this research and without it this work would not come at this quality.

First, I would like to express my deep appreciation to my advisor Prof. Parish on his support, patient, trust, and encouragement. David's guidance was essential to complete this work. His tremendous inspiration and excellent advice led me to deeper understanding of the topic and helped me to grow as a researcher at the field.

I would also like to thank Dr. Peter Sandford who spent time at the initial stage of this work to teach me the simulation ideas and the different approaches of the research methodologies.

I will not forget the people who shared with me their experience of their own work. Dr. Yonggang Zhao from the University of Southern California, Dr. Bhasker Krishnamachari from the University of Southern California, Dr Mehment C. Vuran from Georgia Institute of Technology, Dr. Chih-fan Hsin from the University of Michigan and Mr. Yao-jang Wen from the University of Califorina, Berkeley.

Last and not least I would like to extend my appreciation to my colleagues at Loughborough University High Speed Network Research Group (HNS) for their support and advice during the research, Dr. Omar Bashir, Dr. Mark Sandford, Dr. Mark Withall, Dr. Shirantha De Silva, Dr Momen Al-Rawi, and my best friend Mr. Konstantinos Kyriakopoulos.

Abbreviations and Acronyms

ADC	Analogue-to-Digital Converter
ANN	Artificial Neural Network
API	Application Programming Interface
AQs	Aggregation Queries
BWAN	Broadband Wireless Access Network
CDMA	Code division Multiple Access
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CPU	Central Processing Unit
EEPROM	Electrically Erasable Programmable Read-only Memory
GE	General Electric
GPS	Global Positioning System
GUI	Graphical User Interface
HSN	High Speed Network Research Group
ID	Identification
ISO	International Organization for Standardization
LPC	Linear Predictive Coding
DKF	Discrete Kalman Filter
MAC	Media Access Control
MANNA	Management Architecture for WSNs
MPA	Probabilities approach of maximum a posterior
MATLAB	MATrix LABoratory
nesC	Network Embedded System C
NRL	Naval Research Laboratory (extension to NS for Wireless Sensor Networks)
NS2	Network Simulator 2
NCAP	Network Capable Application Processor
OSI	Open System Interconnection
PCA	Principal Component Analysis

PowerTossim	Power Tiny Microthreading Operating System Simulator
QoS	Quality of Service
RAM	Random Access Memory
RF	Radio Frequency
ROM	Read Only Memory
STIM	Smart Transducer Interface Module
SSQs	Single Source Queries
S-MAC	Sensor Media Access Control
SNAQs	Set of Non-Aggregation Queries
TDMA	Time Division Multiple Access
TEDS	Transducer Electronic Data Sheet
TinyOS	Tiny Microthreading Operating System
TII	Transducer Independent Interface
TOSSIM	Tiny Microthreading Operating System Simulator
TTP/A	Time Triggered System Protocol
WMAV Fusion	Weighted Moving Average Fusion
UART	Universal Asynchronous Receiver and Transmitter
VMBA	Voting Median Base Algorithm
WLAN	Wireless Local area Network
WSN	Wireless Sensor Network
WSNDiag	Wireless Sensor Network Diagnosis

Table of Contents

Abstract

Wireless Sensor Networks (*WSNs*) are expected to be a new, revolutionary technology in the same manner as the Internet. This is due to their special characteristics such as low power consumption, ad hoc operation, self-maintenance and many other features. These special characteristics help in reducing the costs of network manufacture and implementation which extends their applications in a number of areas such as health and military services. Unfortunately, network resources such as memory, power and processing capacity constitute a serious constraint. In addition, they reduce the immunity of the network against external and internal impacts (such as electromagnetic interference) which make sensor node operations frequently deviate from the norm, degrading the *WSN's* functionality. In some cases the data collected by the network becomes unreliable; the monitoring of the phenomenon may even fail. To ensure the reliability of the network, several tools have been proposed to detect and isolate these deviations but most use relatively high levels of resources. In certain circumstances these state-of-the-art tools are unable to avoid the instant impact of data deviations on the accuracy of the collected data and on the network's functionality.

This thesis overcomes these drawbacks by proposing a new, real-time, low resources usage, distributed performance algorithm that will monitor the accuracy of collected data and network functionality in large scale dense deployed *WSNs*. In order to achieve this, we have used the spatio-temporal correlation between the measurements of the neighbour nodes in large scale dense deployed *WSNs*. This correlation arises due to near proximity (of the nodes) and/or the slow characteristics' change of monitored phenomenon.

The proposed algorithm has been tested via simulation experiments using different simulated and real world application data sets. Moreover, it has been tested on a real network testbed with Mote sensors using continuous reporting and event-driven applications. The results from these experiments showed a high rate of detection of changes in the reliability levels of data and in network performance. They also showed a high level of accuracy in terms of the detection of sensor faults. This, however, comes alongside certain limitations because of the use of simple passive analysis with the proposed algorithm.

Chapter 1 Introduction

1.1 Wireless Sensor Networks (*WSNs*)

Wireless sensor networks (*WSNs*) consist of a large number of small wireless, low-power, unattended sensor nodes deployed on a large, ad-hoc scale. Nodes in the network organise themselves after deployment in the field and collaborate with each other in processing and transmitting the collected data in order to ensure highly robust, highly accurate measured phenomena and to reduce the resource usage of network nodes [1]-[5]. The main advantage of this type of network is its ability to be deployed in many kinds of terrain with a hostile environment where it is not possible to use traditional wired networks. This allows the network to extend the virtual functionality of traditional networks to interact directly with the real world allowing network users closely to monitor and control the physical world. Because of this, there is a large number of potential applications and uses for such a network such as surveillance and security, environmental monitoring, transport, precision agriculture, manufacturing and inventory tracking, and health care [6].

1.2 Problem Definition

Nodes in *WSNs* are prone to temporary and permanent deviations from norm as a result of the cheap manufacturing process used to produce them, the material they are made from, the limitations of node resources, and the harsh environment they directly operate in when they measure a phenomenon [7],[8]. Moreover, these deviations increase because of using a wireless medium in communication, the event-driven nature of the operating system and the limited usage of fault-tolerant and diagnosis techniques. As a result, nodes in *WSNs* and their communications are not immune to internal or external impacts, such as changes in environmental conditions and electromagnetic interference, which increase the probability of frequent measurement deviations and node malfunctions.

Since the node in a *WSN* is the basic unit of data creation and communication, any long term deviation or malfunction may badly affects both

the quality and the quantity of the collected data. This degrades the network's functionality in terms of the accuracy of the collected data, data delivery and resource usage. The effect of this deviation/malfunction in the network depends on the degree of deviation, its duration, the protocol's tolerance to this deviation, and the position of the deviated node in the network's collaboration function.

To reduce the effect of node malfunction/deviation on overall network functionality, research studies, such as those in [8]-[10], have proposed several protocols and algorithms that clean/rectify the data and detect/isolate faults such as data cleaning, fault tolerance and diagnosis techniques. The problem with most of these techniques is the large amount of resources they use, the time delay in detection that causes the network's functionality to degrade, and the subsequent reduction in the accuracy of the collected data due to the impact of the deviations on network functionality. Most of these techniques also have relatively complex analyses that tradeoff node resource usage and the accuracy of the analyses. Moreover, they do not detect the degree of degradation of network functionality and the reliability level of collected data in the network. As a result of this, *WSNs* still need methods that indicate when and where this deviation occurs, the type of deviation, and the degree to which it affects network functionality and the reliability of the collected data. Moreover, these methods should have low resource usage so they will not have an impact on the network's lifetime.

Performance monitoring tools offer some of these characteristics; they can overcome the problems previously described as they can detect network functionality degradation before it occurs and before it has a high level of impact on the accuracy of the network's collected data and usage of resources. However, the main problem with such tools is that to implement them directly into an available traditional network is impractical due to their high impact on the *WSN's* lifetime and the lack of availability of the required global parameters for their metric calculation. This creates the need for performance-monitoring algorithms and metrics that would suit *WSNs*.

1.3 Motivation for and Aims of the Thesis

This thesis is motivated by the need to find a method to detect the degradation of Wireless Sensor Network functionality before it has a high level of impact on the accuracy of collected data and network lifetime. Such a method should use a low level of network resources and not have a dramatic effect on the network's lifetime. Moreover, the thesis is also motivated by the need to find new events that will analyse the common metrics used in all *WSN* applications and that will detect the status of changes in the network functionality.

1.4 Challenges

Several challenges must be faced when detecting *WSN* node functionality; the method must distinguish between failed, un-calibrated and real phenomena in measured data. Diverting from other neighbour operation is not always considered to be a node or a communication malfunction but may be due to the effects of the unpredictable nature of the phenomena or environmental factors which, in turn, affect the functioning of nodes and protocols. In addition, there are many challenges that hinder the flexible use of *WSN* performance measurements/monitoring, such as the limited access that nodes have to their non-neighbours' data, the limited capability they have in processing and storing large amounts of data, the high percentage of packet losses in the network, limited power supply, and the high complexity of *WSNs*. This imposes challenges on the simplicity of any proposed performance monitoring/measuring tool and the accuracy of its detection.

To overcome these challenges, a performance algorithm should take into account the characteristics and challenges presented by *WSNs*, such as node constraint resources and the unavailability of dominant protocols or algorithms suitable for all applications. Thus, the proposed algorithm should not have a significant impact on the network's lifetime while monitoring its performance. Moreover, such an algorithm should extract its metrics from parameters that are available in the two levels of network. These two network levels are the

higher level (measured sensed values) and the lower level (network communications values). These two levels effect the networks' data collection and its exchange. Unfortunately, the dependency of the two network levels on each other makes it difficult to separate them while monitoring network performance. For example, the packet losses at a low level impact directly on the reliability measurement at a high network level, as discussed in [11]. Also, it is difficult to gather information from these two levels, relate them to each other, and track the changes from the data initiator node to the destination owing to the types of mechanism used by some routing, Media Access Control (*MAC*) and other protocols that work to prolong the network's lifetime.

1.5 Underline Assumptions for the Research

The algorithm to be described was designed for large-scale densely deployed networks with resource constraints (A multi-hop network with full collaboration between its nodes is considered as a large scale network). It was also designed to be as simple as possible in order for it not to have a high impact on the network's resources or the application functionality while carrying out its monitoring. This was achieved by extracting its parameters from the existing network protocols/characteristics.

The algorithm analysis depends on the similarity of sensed measurements raised between neighbours within dense deployment large scale *WSNs*. This is done by tracking the measurements that are less than the expected phenomenon value at the end of the node receiving range and considering them as correlated. As a result, the algorithm cannot work without a measurement correlation between neighbours in close proximity of each other. (Please note that neighbours are those that can hear each other within their communication range.) This means that measurements over time at neighbouring sensor nodes share an amount of mutual information. This is true in many physical applications, such as applications which monitor an ecological environment, where the phenomenon's characteristics are spread out geographically over the area. In these applications the sensed physical

value, such as temperature or light, shows a degree of spatio-temporal coherence. This correlation increases in the type of large-scale Wireless Sensor Network considered in thesis due to the dense deployment of the nodes. (The dense deployment is done in the network to provide redundancy that reduce the cost of deploying the network, to increase the reliability of the information that is collected, to increase the network's coverage/connectivity and to auto-maintain/auto-configure the network.) This means that if the phenomenon experiences a change in a non-predictable manner within that space (i.e. no correlation), the proposed algorithm will not work because it will have lost the most important factor that it uses to analyse and categorise the functionality of the nodes in the network.

The second important factor that the algorithm relies on for its functionality is the synchronization of the network's communication protocols and the broadcast of packets within the neighbourhood. This is because the algorithm relies for its analysis on the available communication protocols, such as routing protocols and *MAC protocols*, that control the communication in the neighbourhood. Routing protocols, such as the multi-hop routing protocol [15] that is used in testing the proposed algorithm; monitor all traffic received at the node and directly receive update messages that are sent to neighbours within a single hop on a regular basis. This is required to provide link estimation and parent selection mechanisms for ensuring communication between different nodes in the network. In other words, this protocol expects to receive a number of packets within a period of time in order to decide the quality of the link and the stability of the connection between the child and its parents. Another example of communication protocols is *S-MAC* [16]. *S-MAC* is a locally managed synchronization and periodic sleep-listen scheduling protocol. The neighbouring nodes in this protocol form virtual clusters to set up a common sleep schedule. The schedule exchanges are accomplished by periodic synchronization packet broadcasts to immediate neighbours (the period for each node to send a synchronization packet is called the synchronization period).

Using *TinyOs* in *WSNs* allowed a reduction of the time synchronization error between neighbour nodes to less than a *'bit'* time [17], [18]. This is because *TinyOS* allows a component to be interposed deep within the radio stack to signal an event precisely when the first bit of data is transmitted (This eliminates media access delay from calculations). Moreover, the receivers can take a timestamp when they hear the first data bit. The comparison of these timestamps can reduce time synchronization error.

The proposed algorithm assumes the synchronous timing of the communication protocols or application and regular transmission so that, there will be no need for a special synchronous timer or dedicated packet exchange between neighbours in order for the proposed algorithm to function.

The network's deployment goals and network protocols need to be known before network deployment so that a deviation threshold value can be calculated.

Finally, all nodes in the network must have the same characteristics (i.e. they must be homogenous).

1.6 Contribution

The major contribution of this thesis is in developing a real-time distributed algorithm that will monitor large scale dense deployed *WSN* performance while using low node resources. This algorithm requires only the most recent neighbours' measurements and does not rely on any information regarding global topology. The proposed algorithm uses parameters found in nodes for other networking and application protocols; this makes it much cheaper in terms of resource usage. It uses only the transceiver to send warning messages if there is a high level of confidence of network performance degradation or when the node disagrees with the warning messages of neighbours.

The second contribution of this thesis is the events that the algorithm analyses to detect node status. These events relate the high and the low network levels parameters (i.e. measured sensed values and network communications values respectively) and calculate their impact on the accuracy of network collected data and the network's functionality. For high-level parameters it indicates the accuracy value of a group and at low level it indicates the amount of losses and their impact on distorting the collection and communication abilities of neighbour nodes. Other researchers, such as [8],[12]-[14], concentrated on collecting the network performance of either high or low network levels by, for example, tracking energy consumption, connectivity, and coverage. However, this was without taking care to measure both levels to test the tradeoff between them caused by collaborative functions and the effect of these on the accuracy of collected data. For example, if the network design is based on power consumption, the functionality of the network protocols is mainly based on a power consumption tradeoff strategy; such as the reporting rate of nodes or the routing/gathering construction of the data network tree; to control this functionality. Only very little attention is paid to the accuracy of the data collected by the network although this may reduce the quality of the data to a level where they become meaningless to the network's user. On the other hand, if the network design is based on the reliability of the collected data, the network protocols' functionality is mainly based on a data accuracy tradeoff strategy in order to control the network's functionality. In some cases, this may cause an increase in the network's resource usage that will then dramatically reduce the network's expected lifetime.

The third contribution of this study concerns the location of the algorithm functionality in the application source code on the complexity of the proposed algorithm, and the resource usage.

The thesis evaluates the proposed algorithm by a novel combination of the three research methods (i.e. analytical, simulation and empirical). This novel combination of the three research methods provides the fourth contribution of the thesis as discussed in Chapter 9.

1.7 Design and Evaluation of the New Approach

The proposed algorithm was tested using analytical, simulation and empirical experiments to examine its functionality and also its limitations under several scenarios of faults, node deviations and packet losses (as discussed in Chapter 2). These evaluation methods were then compared with each other to test their consistency. Moreover, the experiments were repeated a number of times using random distribution tests, such as losses, node locations and random deviated data, in order to ascertain the confidence interval of the detected results.

These experiments showed that the algorithm achieved a high level of detection reliability regarding network status with low resource usage. In addition, they showed that the proposed algorithm is resilient to both high packet loss and environmental changes, as discussed in Chapters 6, 7 and 8. These results also revealed some limits concerning the algorithm that were detected during the experiments; these are discussed in Chapter 6.

1.8 Implications of Other Research Studies

The algorithm can be useful in many aspects of *WSN* applications. The information collected from the algorithm is helpful in validating the expected sensing functionality or for fine-tuning detection algorithms. It can aid in deploying network nodes to ensure the required deployment goals and can also be helpful in setting network operating parameters such as the setting up of routing tables, node duty cycles, and many others.

The proposed algorithm can improve many routing, *MAC*, dissemination, localisation and aggregation protocols due to its ability to predict suspected malfunctioning nodes. Also, it can improve the network's lifetime and collected data accuracy by isolating suspected malfunctioning nodes, warning network users if there is a high probability of a sudden fault occurrence, or if there is any need for increased network coverage/density.

1.9 Organisation of the Thesis

This thesis is divided into nine chapters. This chapter of the thesis gives an insight into the work. Chapter 2 reviews the basic concepts of Wireless Sensor Networks. It discusses the main components of such networks, how they work at both high and low network levels to achieve application targets, and considers their constraints. Moreover, it discusses different types of recent research in the field. This chapter answers three main questions: what is the gap in the present expertise, why it exists, and what is its importance in terms of a network's lifetime and accuracy.

Chapter 3 explores and introduces network performance measurements, together with the requirements and challenges regarding such measurements in *WSNs*. The chapter discusses different methods and metrics used in measuring and monitoring *WSN* performance, as well as discussing the detection events of the proposed algorithm. Finally, the chapter discusses the related work that has solved network functionality and reliability matters (i.e. data correction and rectification, network diagnosis, fault-tolerance techniques, and *WSN* performance measurement).

Chapter 4 focuses on explaining different modules of the proposed algorithm, their characteristics, the theories behind them, their function, and the events they analyse in order to track the health status of the network. In addition, it discusses methods of controlling the proposed algorithm's detection confidence.

Chapter 5 explains the two implementation methods of the proposed algorithm at protocols stacks, and the modifications to the algorithm that are required to measure the performance of different *WSN* applications.

This is followed by a chapter that evaluates the performance of the algorithm. The chapter also compares the algorithm's functionality with other algorithms and discusses the limitations of the proposed algorithm.

Chapter 7 discusses the simulation results of the three main data sets tested in *MATLAB* code with different faults characteristics; it also discusses three methods for reducing resource usage while the algorithm is working.

This is followed by Chapter 8 which describes the Mica2 Mote nodes that were used in the testbed, the network communication stacks. It also describes the *TinyOS* multi-hop '*Surge*' application that was used to test the proposed algorithm. This chapter discusses the modifications that were made to the main application source code and additional resources used for this modification. Then it summarises the results achieved from the empirical experiments with different topologies and configurations, and different reporting rates. The chapter ends by discussing the different uses of the proposed algorithm.

Finally, Chapter 9 summarises the work that has been completed, the results, and the subsequent conclusions; it also describes the work that might be carried out to extend this research.

Chapter 2 Wireless Sensor Networks

2.1 Introduction

This chapter sets out to explore the knowledge gap that the thesis studies and attempts to bridge. It seeks to establish an understanding of the existing resources as well as constraints and challenges that affect the functionality of Wireless Sensor Networks and the design of protocols. Also, it examines different research in this field, the methodologies and used data sets, and how such research has attempted to dealt with the gap.

2.2 Wireless Sensor Networks

A Wireless Sensor Network (*WSN*) can be defined as a network that consists of a large number of small wireless, low-power and unattended sensors deployed in a large-scale, ad-hoc fashion. This network is used as a tool for measuring and transferring information concerning sensed phenomena to a sink; i.e. data collecting point; within certain requirements, such as level of accuracy, latency and network lifetime [1], [2].

Nodes in this network self-configure and organise themselves after random or planned deployment in the field. They collaborate with each other in sensing the appropriate phenomenon and in processing the collected data in-network to ensure that information concerning the measured phenomenon is highly robust and highly accurate. These nodes generate small packets containing the measurements gathered from the phenomenon and send them to the destination using an underlying routing protocol. This may directly forward the packet to the next node, delay the packet and merge it into one large packet, or extract the data in each packet and aggregate them into a new result which is then forwarded to the destination [1], [2].

In order to reduce the total cost of implementation and maintenance, the network design depends on each application having its own sensor characteristics, sensor deployment, performance metrics, and protocol requirements [3], [19]. As a result of these characteristics, *WSNs* are used in a large number of potential applications with different configurations and

protocols, such as surveillance and security, environmental monitoring, transport, precision agriculture, manufacturing, inventory tracking, and health care [1], [2], [6].

WSN components can be divided into three main parts: sensor nodes, protocols and sinks. Each of them has its own function and effect on the network's functionality and on the collected information.

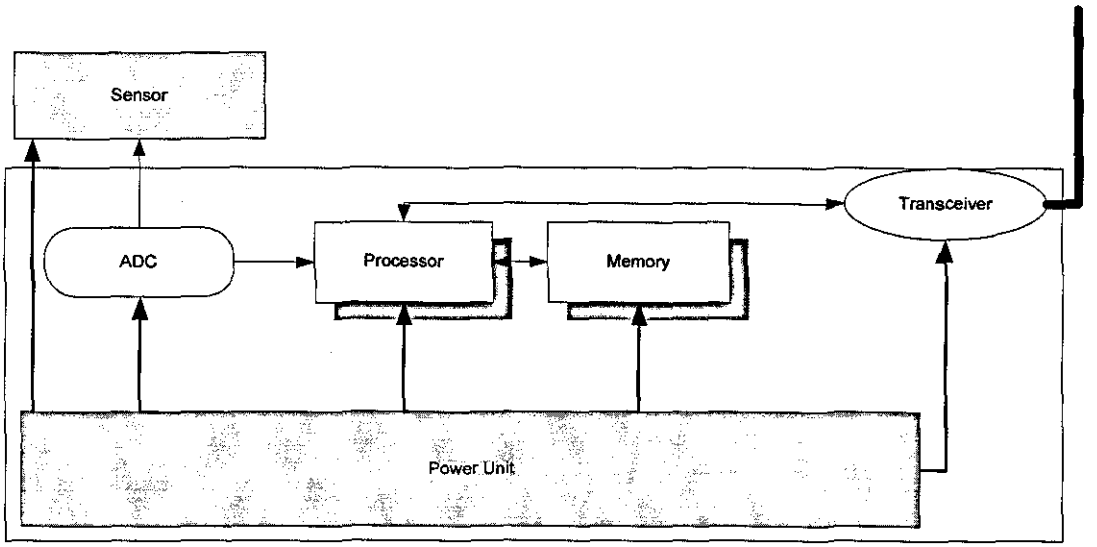

Figure 2-1. Different Wireless Sensor Node Components

2.3 Sensor Nodes

IEEE 1451 Smart Sensor Network Standard nodes in a *WSN* is one of the standards that most of sensor designers relies on. It divides the sensor node into four main components: Smart Transducer Interface Module (*STIM*), Transducer Electronic Data Sheet (*TEDS*), Transducer Independent Interface (*TII*), and Network Capable Application Processor (*NCAP*) [1]. With this standard each wireless sensor node has one or more sensor element devices that can observe or control the physical parameters of the phenomenon; a power supply; memory to store programs and intermediate data; a processor to process all the relevant data and to execute arbitrary code; an analog to digital converter, and a transceiver (as shown in Figure 2-1). Moreover, these nodes can have additional components, such as Global Positioning System (*GPS*), actuator, scavenger and many others, depending on the application.

The size/cost of the sensor node and the characteristics of its components influence the application requirements which also govern the deployment of network nodes, and the functions and complexity of the protocol [19]. Each of these node components has to operate by balancing a tradeoff between the energy and task accuracy specified by the application. The characteristics of node components affect the accuracy of data collected at the sink and the way the network communicates. For example, the memory size at the node affects the buffering space and the ability of the network to handle traffic; it also affects the level of protocol complexity that can be handled by the node. Furthermore, the battery size affects the lifetime of the network; the processing capability determines the level of optimisation that is possible in the node and the complexity of protocols that can operate at the node; and the transceiver determines the transmission range of the network, the capacity of the transmission channel, and the network's stability [3].

These node characteristics have a direct impact on network performance since a sensor node not only works as the source of data but also participates in the network's communication and collaboration. Thus, each node has its own impact on *WSN* performance and the degree to which it affects this performance depends on its location and application characteristics, a factor this study depends on for monitoring *WSN* performance. This is different from most *WSN* research studies, such as [20], [21], as these concentrated on group functionality and the collected data at the sink to detect any change. These researchers built their argument on the low impact of these individual nodes on network reliability as a result of the large amount of correlated data near nodes which reduces the effect of individual node deviation. However, this constructed group (depended on by these researchers) is based on the degree of common characteristics among neighbour nodes. If these common characteristics change the group size, the group configuration and the group usage of resources also change. This change the premise that such research depends on.

In general sensor nodes can be divided into four main types depending on their functionality and resource capacity:

- Common nodes: responsible for collecting sensing data.
- Sink nodes: responsible for collecting, storing and processing data collected from the common nodes using static or dynamic queries.
- Gateway nodes: Those connect sink nodes to an external network.
- Actuators: which offer control of or actuation in the monitored area.

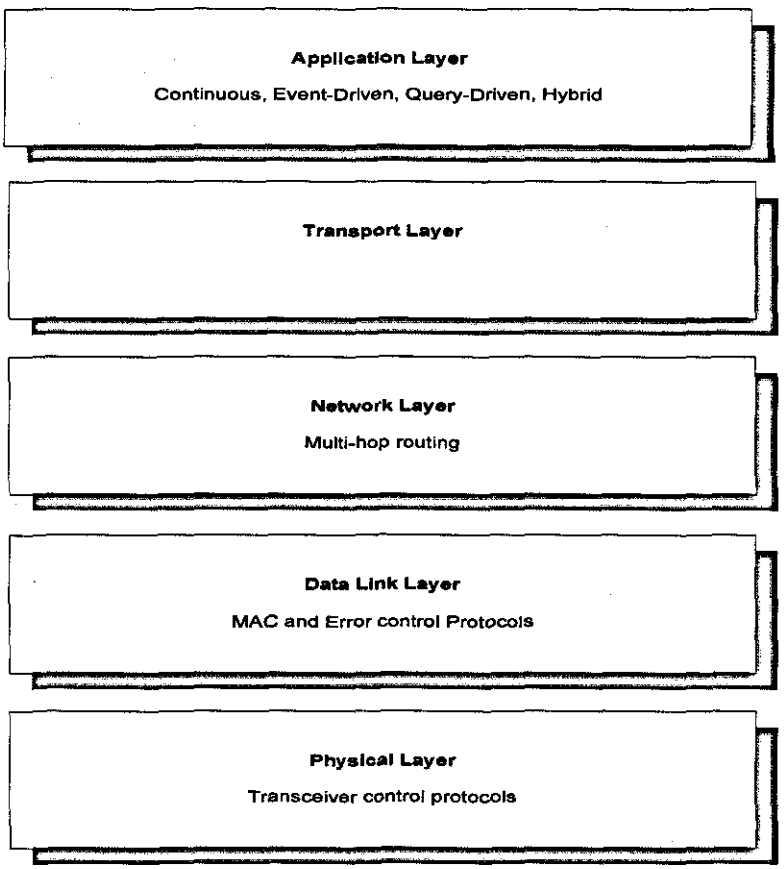

Figure 2-2. *WSN* Protocol Stack [2]

2.4 Protocols

The architectures of *WSNs* vary in their complexity as a result of different application requirements in terms of latency, accuracy and network lifetimes. Each of these applications specifies protocols that reduce the complexity of an application's functions by using in-network, distributed processing tasks at different protocol stack levels, as shown in Figure 2-2 (Note: The transport layer in the *WSN* protocol stack is used when it is desired to access WSN through the Internet or some external networks. In this case the network layer

is used by the nodes at the access points between the two networks. This means that not all wireless sensor nodes have every protocol layer).

The design of *WSN* protocols is based on maximising a network's lifetime by managing its energy in an efficient manner (but tradeoff network parameters may cause degradation in the networks' performance and a reduction in the accuracy of the collected data). This, in turn, leads to the lack of availability of a common protocol that will suit all applications. To develop such a protocol is not feasible because of its need to use more resources at the existing platform.

WSNs protocols can be divided into two main groups: applications and communication/infrastructure protocols [19]. The second type followed the first in terms of functionality in areas such as nodes' sleep time, routing table setting, and data collection. The relationship between the two groups creates virtual network functionality trees that save power and also gather/route data.

Application protocols are concerned with the transfer of sensed data to the sink by collaboration/non-collaboration with other nodes and in-network sensed data processes. These protocols are generally divided into four main groups, depending on their data delivery methods [8], [19], [20]:

1) Continuous applications where sensors communicate their data continuously at a pre-specified rate.
2) Event-driven applications where sensors report information only if an event of interest occurs.
3) Observer-initiated applications where sensors only report their results in response to an explicit request from the sink. These applications are divided into Single Source Queries (*SSQs*) that return the values of the sensors in attributes without aggregation; Set of Non-Aggregate Queries (*SNAQs*) that return the set of sensors that satisfy a given user-defined predicate; and Aggregate Queries (*AQs*) that combine a number of messages into a smaller representation that is equivalent.

4) Hybrid applications that are a combination of two or three of these types.

Communication/infrastructure protocols, such as dissemination, route and *MAC* protocols, are responsible for configuring, maintaining and optimising operations to keep the network functioning and to ensure robust operation in dynamic environments, as well as optimising overall performance [19]. These protocols influence the amount of communication required in a network, where the initial phase of infrastructure communication is needed to set up the network and there will be additional communication for reconfiguration, which is used for network optimisation. These are different from traditional network protocols in their special attention, which then is traded off in terms of the energy and network deployment goals, as discussed in [5],[16], [19], [22].

Both application and communication protocols are basically divided into:

- Table-driven protocols (i.e. proactive protocol): Those maintain up-to-date information from each node to all other nodes through the routing table.
- On-demand protocols (i.e. reactive protocols): where a node requests a discovery process to establish a route and maintain the route as required; i.e. they are used mostly in query applications such as flooding or information-driven routing protocols.

2.5 Research and Challenges in the Field

Wireless Sensor Networks have increasingly attracted research interest, given recent advances in design regarding minimisation, low cost and low power. The necessity for energy efficiency governs all aspects of system design due to the need for applications to use limited energy resources and because there is limited potential for changing most applications after they have been deployed. Researchers focusing their attentions on the field of *WSNs* are

attempting to maximise the utilisation of these networks with high scalability, high levels of accuracy in collected data, while minimising energy consumption. This target, achieved by [1], [2], [20]-[22]:

- Tradeoff between communication and processing. This depends on data properties, node densities and environmental circumstances.
- Tradeoff between energy and network protocols in network discovery, maintenance and tracking using proactive routing, reactive routing, and hybrid solutions.
- MAC layer tradeoff and power-saving techniques such as node sleeping.
- Tradeoff between collected data accuracy, sampling rates, reporting rates and energy consumption.

The problem surrounding these tradeoffs is that they only work if there is an effect on the network's functionality that, in some cases, may causes a large usage of resources or a disconnection in communication links. When this happens, a sudden degradation in network functionality may occurs and, until this is resolved, it reduces performance and consumes resources. Also, these tradeoffs are responsible for balancing, as far as possible, network Quality of Service (QoS) and resource usage in order to prolong the network's lifetime without alarming network users with the level of degradation in the accuracy of the collected data as a result of the tradeoff.

2.6 Functionality and Deviation Measurements of Wireless Sensor Networks

Deviations in sensor node operations arise as a result of systematic or transient errors [8]. These two types of error directly and indirectly affect the quality and the quantity of data collected by the *WSN*. They directly affect sensor measurements and cause frequent deviations from the real value of the monitored phenomena, as discussed in [8], [13], [23]. Furthermore, if there are a large variety of deviations, they may affect the quality and quantity of the

network's collected data. This is because node measurements in *WSNs* are not only important due to their values reflect characteristics of the monitored phenomena, but they also control the functionality of the network especially in a large-scale deployment; where network nodes self-organise and self-configure themselves by deciding on collaboration, aggregation, communication operations, control routing paths, data collection points, and data coding [24],[25]. So, any lengthy deviation of node measurements may cause variations in the performance of the node protocols, change the percentage of detection accuracy, vary the usage of node resources and, in the end, this may degrade the network's performance if its time or/and value is larger enough [26]. For example, communication errors lead to transmission overheads of between 40-160% [27].

In general, the sensor node measurement value can be expressed as:

$$M(t) = O(t) + G(t) * P(t) + N(t) \qquad (2.1)$$

where *M(t)* is the sensor node measurement value, *O(t)* is the sensor offset that arises as a result of calibration error, *G(t)* is the sensor gain that happens as a result of the sensor response to changes in the phenomenon, *P(t)* the real value of the phenomenon, and *N(t)* is the noise that arises as a result of the effects of external sources on the hardware. The values of these parameters produce two types of error (i.e. permanent and transient errors [8], [26], [27].).

The permanent type arise as a result of a permanent change in node behaviour; (this is called systematic error). They affect node functionality continuously until the problem is rectified and are caused mainly by hardware faults, such as a calibration error after prolonged use or a reduction in operating power levels. Researchers have solved this type of deviation by providing reliability against communication failure at the transport level, ad-hoc routing techniques that can cope with dynamic failure, and data aggregation and fusion in the network.

The other type arises as a result of an intermittent or transient deviation from normal behaviour (called random/noise error). This occurs as a result of temporary external or internal conditions such as various random environmental effects, unstable characteristics of the hardware, software bugs, or channel interface and multi-path effects. The functionality of the nodes in such a fault returns back to normal after the effect disappears. Researchers have solved this problem by using enhancement techniques for circuit-level, channel-coding techniques, and by using aggregation and fusion techniques.

Unfortunately, these existing solutions for both error type use up high levels of resources in the sensor nodes in terms of energy, bandwidth and cost overheads. They also take time until they rectify the cause which may degrade the network's functionality, reduce the accuracy of the collected data, and consume more resources.

On the other hand, the measurement errors mentioned above can have an indirect effect on the network's collaboration function, the construction of routing tables, the selection of the node reporting rate, and the selection of data gathering points since the network uses neighbourhood measurements for its functionality decisions between neighbour nodes.

2.7 Performance Monitoring Gap in Wireless Sensor Networks

In order for unattended *WSNs* to operate at a satisfactory level for a long time, they need to be able to adapt to changes in wireless communication, environmental changes, failures, workload changes, and power reductions. This can only be done if the network has self-healing, self-monitoring, self-organising, self-managing, self-calibration, and self-power management protocols. For this, researchers used fault-tolerance techniques such as error detection and correction in localisation, routing and *MAC* protocols; such as in [16].

Although all these techniques try to reconfigure the network communication to ensure network connectivity and low power consumption, given a sudden change they may use high levels of resources until the network reconfigures itself again. Moreover, most of these techniques track crash faults of either network or information/data (such as node measurements and packet loss) without indicating their effect(s) on each other. Crash faults can be defined as, "the miscellaneous behavior of the operation". A crash fault can occur in any of the three operations of sensor nodes. In the communication operation a crash fault occurs when the nodes cannot communicate. In its sensing operation, a crash fault occurs when the sensed value (of the monitored phenomenon) deviates significantly from its actual value. In the operation of processing, a crash fault is said to occur when node perform wrong operation due to its analysis.

Performance monitoring techniques can overcome these problems by using network and/or data/information values for online passive monitoring. These types of technique are useful when analysing the problems in terms of characterising their nature and their impact on the network so that the network will not have its functionality seriously degraded. This, however, is an unexplored research field because of the special characteristics of *WSNs.* These characteristics include resource constraints, different network configuration and protocol settings for each application, and the network's dependency on collaboration between a large number of nodes. These complicate research studies and make it not entirely clear how to describe *WSN* services or how to relate the two network levels to each other.

2.8 Methodology of the Research

As with traditional networks, research into Wireless Sensor Networks generates new protocols/algorithms using analytical methods, computer simulations, and/or empirical experiments.

Analytical techniques study the trade-offs in *WSN* parameters from a mathematical point of view and these techniques help designers to consider,

during the design process, all the trade-offs involved. This consideration of the trade-offs is achieved by developing a system model that contains the parts of the actual system related to the analysis. The model's analysis mechanism may utilise probability functions, graph algorithms, non-deterministic polynomial-time hard problems (*NP-hard),* and Markov chains. The eight main popular analytical techniques used with *WSNs* are shown in Table 2- 1[28] below.

Technique	Main goal	Level of analysis
Target detection performance analysis	Finds the probability of detection of a target by any sensor node: e.g. [29].	Application level and *WSN* level
Communication/data management analysis	Finds out if a schedule for multi-hop end-to-end communication streams is able to fulfill deadlines of all the streams in the system: e.g. [30].	System level and communication level
Sensing coverage	Determines how well each point in the given sensing area is covered by *WSN*: e.g. [31].	System level and communication level
Capacity analysis	Derives information theoretic for *WSN* that quantifies the ability of the network to transfer data across distances; e.g. [32].	System level and communication level
Reliability analysis	Considers sensor capabilities, the number of sensor nodes, and deployment strategies: e.g. [33].	System level and node level
Connectivity analysis	Determines whether the WSN system is connected or not. The connectivity analysis may also include deriving the relation for energy consumption needed to maintain the desired level in the network: e.g. [34].	System level, communication level and node level
Security-related analysis	Evaluates different security schemes for a WSN or decides the effectiveness of a security scheme against certain types of attack: e.g. [35].	System level and communication level
Lifetime analysis	Finds out when a *WSN* or individual sensor node is expected to run out of energy: e.g. [36].	System level and node level

Table 2-1. The Main Categories of Analytical Models for *WSNs*

Unfortunately, in Wireless Sensor Networks there are many aspects to consider; such as energy efficiency, the limited resources of nodes, decentralised collaboration between different nodes, fault tolerance, and the global behaviour emerging; from local interaction, that can create unpredictable network behaviours. These unpredictable behaviours increase

the difficulties and the errors associated with constructing analytical models and therefore need to be considered when producing an analytical model.

The computer simulation method quickly and inexpensively explores the behaviour of protocols and algorithms across several scenarios, such as topologies and traffic [37], [38]. It uses many linked tools to simulate a multitude of parameters; it is also easy to use and control, cheap to implement and can take a large number of nodes. It contains a rich infrastructure for developing new protocols, offers good opportunities to study large-scale protocol interactions in a controlled environment, and makes the comparison of results across research efforts easier. However, in order to evaluate performance, accurate sensing, communication and collaboration modules should be programmed. This increases the difficulty of designing a complicated simulation tool for individual sensor nodes in a *WSN*. There are many simulations which can be used to examine sensor network behaviour, as shown in Table 2.2, and each of these has a particular area of expertise in which it excels.

Most of the research studies, such as [12], [14], [39]-[41], have used computer simulations and analytical methods to test the performance of their proposed algorithms and protocols. However, the behaviour of Wireless Sensor Network algorithms tends to be complex and very difficult to predict using analytical or simulation methods. This is because it is difficult, with these two methods, to represent practical faults and their environmental impact since several aspects are difficult to model. These include aspects such as energy efficiency, limited node resources, decentralised collaboration behaviour between different nodes, the effect of the monitored phenomenon on nodes in the network, network dynamics, and wireless behaviour.

Finally, an empirical method shows the actual implementation of the algorithm in the real world and its sensitivity to different factors. Nonetheless, there are also several limitations with this method such as the fact that the experiment may be conducted on a small-scale evaluation testbed due to cost and the impracticality of carrying out an evaluation on a large network. In addition,

this method lacks a wide traffic mix and the variety of topologies found in real networks; difficulties also exist regarding programming, repeating and controlling the different factors concerning the measurements that need to be tested.

Simulator	Simulation Model	Languages	Description
NS2 [42]	*ISO/OSI*	OTCL, C++	Includes a huge number of protocols, traffic generators and tools to simulate TCP, routing, and multicast protocols over wired and wireless networks.
NRL's sensor extension to NS2 [43]	*ISO/OSI*	OTCL, C++, (Object-Oriented)	Models the presence of a phenomenon transmitted through a designated channel in NS2.
OMNeT++[44]	*ISO/OSI*	C++(Object-oriented)	Is a component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel.
GloMoSiM [45]	*ISO/OSI*	C++, (Object-Oriented)	Is a standard API used between the different simulation layers. The simulation is built on top of Parsec.
SENSE[46]	*ISO/OSI*	C++ (component-port model)	Offers different battery models, simple network and application layers, and an IEEE 802.11 implementation.
TOSSIM [47]	At bit level	*NesC*	Simulates TinyOS motes.
OPNET [48]	*ISO/OSI*	*C/C++*	Provides a simulation language with network libraries.
MATLAB [49]	----	M-code	Is a Numerical computing environment and programming language, created by The MathWorks, MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages.

Table 2-2. Some Wireless Sensor Networks Simulators.

2.8.1 Methodologies Used in the Research

From the above discussion, it can be concluded that the most feasible approach to test the algorithm in *WSNs*, and the method that has been adopted in this thesis, is a combination of the three methods mentioned above. This is to ensure the functionality of the algorithm and to overcome the drawbacks of each separate, stand-alone method. This is discussed in Chapters 6, 7 and 8.

MATLAB, *TOSIMM*, and *PowerTossim* were used in the simulation experiments. *NS2* and its extension *NRL* were tested for two months but their functionality depended on simulating the network level and the ad hoc nature of delivered packets. Moreover, the event-driven nature of the practical implementation code of *WSNs* is not simulated in these simulators. The functionality of the proposed algorithm needed to be evaluated at an application level and the impact of the network level on the received data, collaboration functionality, and errors regarding in-network analysis needed to be tested; which is not possible in *NS2* or its extension *NRL*. This was done in *MATLAB* by modeling the functionality of the algorithm at node and network levels.

The simulation experiments that were conducted in *MATLAB* can be divided into two groups. The first experiments simulated the effect of temporarily deviated data, packet losses and permanently deviated data on the algorithm detection performance in the network. The second simulated the effect of variations in temporarily deviated data, packet losses and permanently deviated data in the node. This was carried out in terms of changes in residual growth, the reliability of neighbourhood readings, and the reliability of neighbourhood communications.

The simulation experiments were repeated either 33 or 100 times depending on variations in the tested data. This is to ensure the confidence level of the detected results [51].

An analytically simple Bayesian method [50] was used to model the algorithm detection performance using the probability of neighbours having a measurement within the threshold and the probability that a fault is above the threshold. This method was used to model these detection scenarios because it is simple and there is no need to consider any phenomenon or equipment.

Finally, the empirical experiments were carried out using a Crossbow Mote testbed and were conducted for one-hop and multi-hop operation. The one-hop experiments tested the algorithm's detection of the warning messages

released in the neighbourhood by varying the number of faults, by increasing packet losses, and by changing the threshold values. The multi-hop operation tested the effect of packet losses and high dynamic topology on the detection of deviated and dead nodes.

Moreover, the consistency of the three methods was tested by comparing the detection of the number of deviated nodes in 10 node neighbourhoods.(Please note that the number of nodes chosen in the neighbourhood was based on the number of nodes that were available in the testbed.)

2.8.2 Evaluation of the Algorithm' Performance

The three methods were used to test various scenarios in order to evaluate the performance of the algorithm in three main areas: resource usage, estimated value for the phenomenon, and detection performance.

The first area tested was the usage of resources. This is because resource constraints directly affect a network's lifetime. Three main tests were used for this evaluation. First was the algorithm analysis technique [52] and this was used to estimate theoretically the memory usage, the running time required, and the exchange packets for both best and worst case scenarios. The second test compared running time and the memory storage of the median calculation used in the algorithm with a statistical redundancy method (i.e. an average), a fusion approach [53], and analytical redundancy techniques [54]. These methods were selected because of their low level of resource usage and their high efficiency in estimating phenomenon values. This was followed by measuring the power consumption of the algorithm using *MATLAB* code and the power model given in Appendix A. This was done to check energy consumption at node level as a result of the release of warning messages. Moreover, several experiments were conducted using *PowerTossim* to test the *CPU* consumption of nodes with and without the addition of the algorithm. In addition, the impact on the lifetime of network nodes was calculated using a maximum number of sent and received warning packets with different

neighbourhood sizes. (This evaluation was carried out based on the *TinyOs* 'Surge' application.) Finally, the algorithm power consumption is compared with the centralized and aggregation methods that collect the individual status of network nodes and send them to the sink for further analysis. (Routing to the sink used a combination of a distributed election leader and a distance-vector, as described in [23].)

The second area tested was the estimated value error. This was done by calculating the absolute error in the algorithm's estimated value of the phenomenon and other algorithm's estimations; this value is important because the algorithm analysis depends on it. The absolute error was calculated between the median and estimators (i.e. *TTP/A WMAV* [53], second order linear prediction [55], and Gaussian correlation technique). These estimators were used because of their robustness and their use in some *WSN* applications.

The third aspect of evaluation tested the detection performance and this was achieved by developing a simple analytical model (a Bayesian model) to assess the detection using different threshold values, different numbers of nodes, and different threshold false probabilities. Moreover, statistical methods for detecting outliers were also used (i.e. Box-whisker and statistical Gaussian limits methods [51]) to compare their detection with that of the algorithm. Also, the detection performance was compared with the Bayesian fault-recognition algorithm (the method used to detect faults in *WSNs*) [23] and with a collaboration fault detection algorithm [56]. Finally, both the positive and negative false detections made by the algorithm were tested in terms of a predefined fault percentage in the neighbourhood.

2.8.3 Data Set Characteristics

As Yan showed in [25], the data that are input into a protocol/algorithm vary its functionality. As a result, the simulation experiments used different data sets with different measurement characteristics and different packet loss charactristics. Data sets can basically be divided into two main groups: real

data sets (i.e. Intel LAB data [57], Botanic Garden Data [58]), and simulated data.

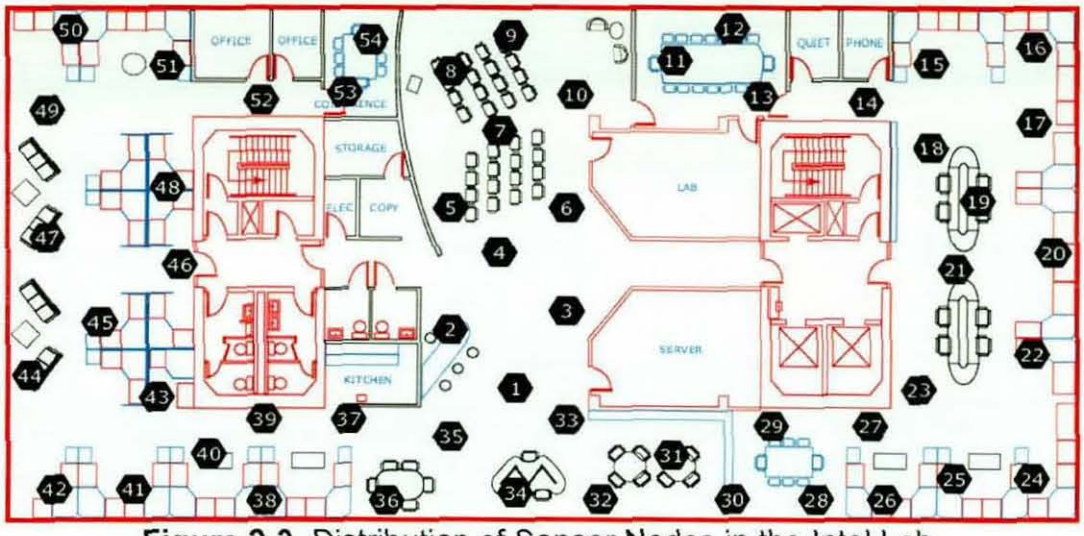

Figure 2-3. Distribution of Sensor Nodes in the Intel Lab.

2.8.3.1 Intel LAB Data Set

The goal of the Intel LAB experiment [57] was to test the behaviour of a sensor network with different conditions of battery power depletion, traffic generation, and multi-hop aspects. In this experiment, 54 nodes were deployed in the Intel lab from February 28th until April 5th 2004 (as shown in Figure 2-3) for 720 hours. A scheduled communication approach was used with a waking period of four seconds and a 13% duty cycle.

The data set collected from this experiment had a lot of missing data (65% of total data was missing), noise (as shown in Figure 2-4 where the measurement of each node deviated from another in either correlated or uncorrelated ways and direction), and failed sensors, especially when the battery level was low at the end of the experiment (i.e. all defective nodes deviated to a high reading before they became dead). As a result of the large number of outliers which were caused by network losses, synchronisation losses and sensor failures, this data set was used to test the robustness of the proposed algorithm under worst real-world conditions. Losses in the data

were reduced to 38% using linear regression in order to produce a second data set for comparison.

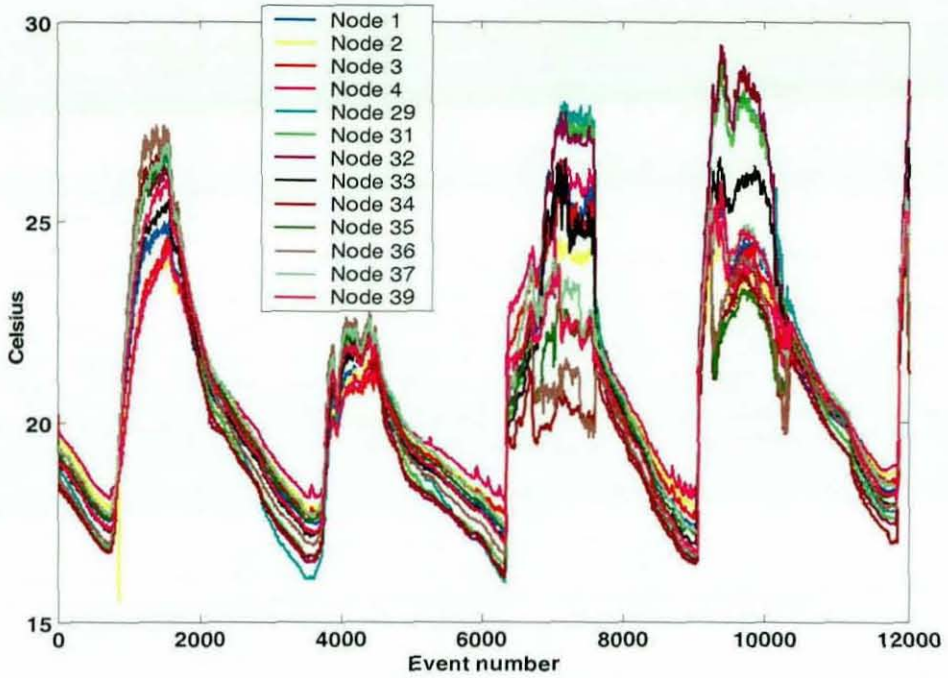

Figure 2-4. Readings from Nodes 1 to 13 in the Intel Lab Experiment between Event 1 and 12000

The neighbours in these experiments were calculated depending on the physical distance between nodes and nodes that were closer to each other than the communication range (i.e. 50 metres) were considered as neighbours. Table B.1 in Appendix B shows the neighbours of Node 1 and the losses among them due to synchronisation, environment effects, node faults and wireless network medium congestion. From the table, it can be seen that there are around 65% data losses (38% of which are due to synchronisation). This statistic for losses changes if calculations are taken from another node such as Node 2 (the experiments were repeated for this node). As shown in Table C.1, because of a high level of loss, cases where no reading at all was received from any neighbour reached 15% (i.e. 13364 events); therefore, with this, the percentage of error in the algorithm increased.

2.8.3.2 Botanic Garden Data Set

The Botanic Garden experiment [58] consisted of placing 11 sensors in a 70m Redwood tree at 4 different altitudes (as shown at Figure 2-5) at the UC

Berkeley Botanical Garden in July 2003 for 20 days. The main goal of this was to measure how environmental parameters varied through the day, outdoors and at different heights. These sensors collected light, humidity, temperature and voltage readings once every 15 minutes (i.e. a total of around 5482 readings) and sent them to a sink 100 metres away.

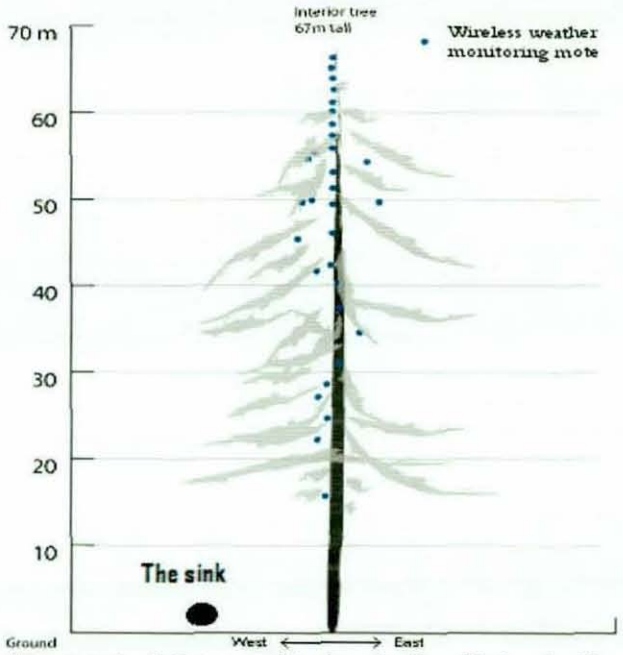

Figure 2-5. Deployment of Sensor Nodes in the Botanic Garden Experiment

This data set has a low level of losses, was a single-hop network and had very few outliers. However, it also contains temporary changes as a result of the impact of outdoor environmental changes on the sensor nodes and has different coverage due to the altitude of each group of sensors.

2.8.3.3 Simulated Data Sets

Simulated data were created to check the proposed algorithm's deviation detection ability at both network and node levels. This was done by using two types of simulated data. The first type was created to test the detection performance of the algorithm at the network level, together with the effect of random deviated data, packet losses, dead nodes and faulty deviated nodes on the algorithm functionality. With this simulated data set, the measurement values of sensor nodes were considered to be either values of '0' to indicate

packet loss, '1' to indicate normal measurement, and '2' to indicate a deviated measurement. Each data set consisted of 100 events with a random or predefined percentage of scenarios generated by a random function generator. The generator selects at what event a sensor deviation occurs, the type of sensor deviation (i.e. permanent or temporary), the deviation duration, and which sensor in the network is deviated.

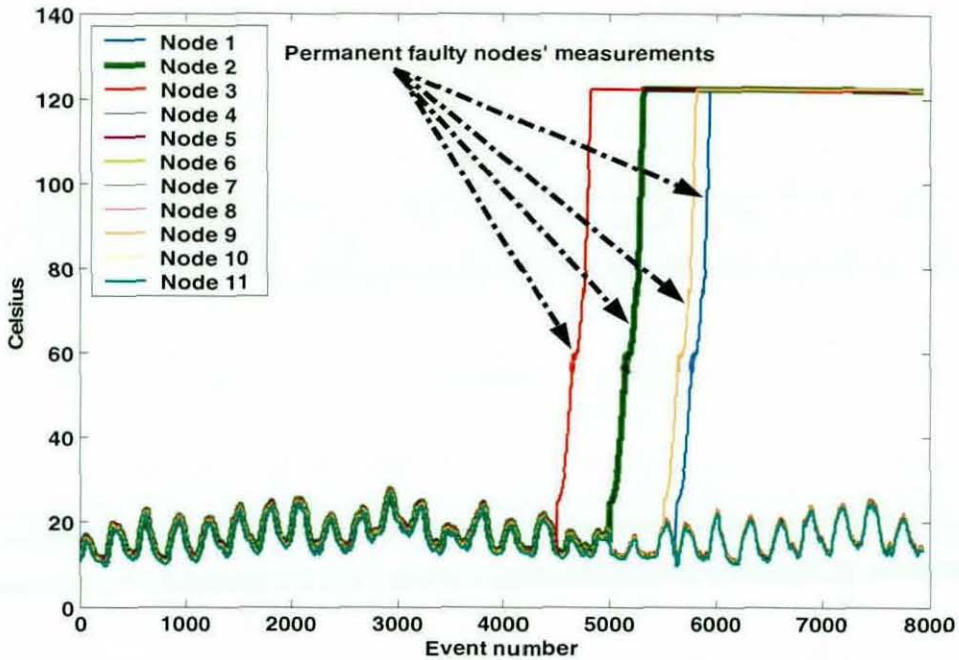

Figure 2-6. The Simulated Data Series for 11 Node Measurements Produced from the Botanic Garden Experiment Data Set

The second simulated data set used a sample of 11 real world temperature measurement patterns taken over 48 hours in the UC Berkley Botanic Garden [58] to model the correlation between different nodes. Then, different scenarios of random data deviations, permanent node deviations, packet losses and dead nodes were superimposed over the original data in the data set to modify it according to predefined percentages, nodes and events, as shown in Figure 2-6. The aim of this data set is to evaluate the performance of the special-temporal algorithm detection by testing the impact of different scenarios in terms of random deviated sensor readings, neighbour packet losses, different sizes of monitoring window and thresholds. It also evaluates the effect of removing the confirmed deviated node on the accuracy of the

neighbourhood collected data, the network performance, and the algorithm detection performance at the node level.

2.9 Summary

This chapter argues that there are many factors that influence *WSN* functionality at high and low network levels as a result of node characteristics, external effects, and the use of protocol tradeoffs and collaboration functions. Such impacts emerge in the node measurements and network collected data. This been solved by proposing different data cleaning, fault-tolerant and diagnosis techniques. Unfortunately, most of these proposed techniques track only one network level while the untracked network level parameters may still affect the functionality of the network by reducing the accuracy of its collected data or by reducing its expected lifetime. The chapter ends by discussing different evaluation methods used in the research.

Chapter 3 Network Performance

3.1 Introduction

This chapter seeks to establish an understanding of traditional performance measurement/monitoring techniques, the challenges that oppose their functioning in *WSNs*, and systematic ways of choosing their metrics and locations in *WSNs*. It also proposes a new *WSN* performance algorithm and desires events which have been analysed through simple application metrics. These events cover most of the previous metrics, such as coverage, connectivity and accuracy, used in other research studies, and predict node malfunctions before they occur by tracking node operation changes in neighbour nodes. Finally, the chapter discusses different research in the field, the methodologies adopted and their limitations.

3.2 Traditional Network Performance Measurements and Monitoring

Performance measurements in large telecommunication networks can be defined as the overall effectiveness of a network at a given time. They are applied to maintain the network's functionality at an acceptable level and to locate network problems especially by finding trends that are not discovered during normal network tests (i.e. fault management, security management, diagnosis tests and configuration management); they can only be located through monitoring over time. These measurements use a set of techniques that implement quantifiable indicators, such as bandwidth, delay/jitter and packet losses, that have a direct impact on data exchange in traditional networks where indicator changes cause a change in the network's functionality. These quantifiable indicators change during a network's lifetime due to changes in usage of shared network resources or/and faults occurring within a network's components as a result of internal or external conditions. The percentage of indicator changes depends on the application's tolerance and the level of Quality of Service (*QoS*) required at the destination.

These performance measurements operate in three main steps [59]. First, performance parameters are gathered by active and/or passive methods. The

active method generates traffic to obtain the required test operation. It has potentially a high level of traffic generated in the monitored network and makes the controller a single point of failure. This is because the monitored node (in active method) sends continuous messages concerning the required parameters to a control unit. The passive method, on the other hand, does not generate traffic but monitors the operation using the available/exchange parameters. Second, these collected data are analysed to determine the normal level. Finally, the analysis results are compared with performance threshold to detect the network problem.

In general, wireless networks performance measurements are divided into two main groups, depending on the network type [59]:

1) Infrastructure-based (such as *WLAN* and *BWAN*) where all mobile hosts in communication reach a base station in one hop. Performance challenges in this context mainly arise from the scaring of bandwidth and the complexity of user mobility during the last wireless hop. These types of network use wired mechanisms, as shown in Table C.1 in Appendix C, but can, with a little modification, tolerate scarce bandwidth and complex mobility.
2) Ad hoc networks where wired network performance measurements cannot be directly implemented because of the bandwidth constraints, the mobility and the dynamic network topology.

3.3 Challenges Regarding Wireless Sensor Network Performance Measurements

Performance measurements in *WSNs* are different from those in large telecommunication networks (summarised above) because the main function of traditional networks is to generate independent information in nodes and exchange them with the destination. However, in *WSNs*, these nodes collaborate with each other in collecting and communicating network data. Moreover, resource constraints with *WSNs* mean that traditional network performance measurement techniques do not have the same flexibility. (Table

3.1 shows some of the differences between traditional networks and *WSNs*.) This makes the usage of traditional network performance measurements in *WSNs* either expensive in terms of resource usage or not possible due to the lack of availability of the required network global parameters. As a result, there is a need for new performance algorithms that measure and monitor *WSN* functionality.

Metric	*WSN*	Traditional network
Intelligence location	Network intelligence depends on the application.	Network intelligence is at the middle between sender and receiver.
Node address	Depending on the application, node address may or may not be required.	Addresses are required for each node.
Type of protocol parameters	Most of the protocols use the local nodes' knowledge in their decision.	Most of the protocols use the network global knowledge in their decision.
Protocol overheads	Low protocol overheads.	Protocols can use high overheads.
Structure	Most are ad hoc structures.	Most are infrastructures.
Packet loss sensitivity	Packet losses can be tolerated in some applications due to redundancy.	Packet losses are one of the critical parameters of network performance.
Criticality of transmitted data/ information	Most of the transmitted information from the monitored area is critical especially in tracking applications.	Transmitted data criticality depends on the application and QoS used for different categories.

Table 3-1. Some Differences between Traditional Networks and *WSNs*

These new performance methods must pay special attention to a network's characteristics that may reduce the monitoring method's detection confidence or that may have a significant impact on the network's lifetime, including [1],[2]:

- Limited and finite energy and communication resources.
- The unavailability of dominant protocols or algorithms that can be used in all applications as a result of the network design's dependency on the application.

- The unavailability of global measurement variables, due to the use of distributed control protocols that reduce the consumption of the network's resources.
- The adaptive nature of networks leads to frequent changes in connectivity, link failure and node status.
- Network traffic becomes unbalanced because of different data characteristics in terms of changes in monitored phenomena, reporting rate duration, and the number of sinks in the network.
- The network's degree of tolerance to data changes and losses as a result of redundancy.
- The collaboration functions used in different tasks to increase the accuracy of collected data and to reduce the usage of node resources.
- The direct interaction of nodes with the environment increases noise levels and increases the probability of node failure.
- Observed fluctuations in phenomena impact on the traffic load and performance of the network.
- Node identification may be unavailable or may have a unique address in some applications.

3.4 Performance Metrics Used in *WSNs*

As discussed in Chapter 2, *WSN* characteristics depend on the type of application that controls the design factors and goals of a network. These factors and goals decide the type of deployment (i.e. node density, node location and the expected degree of network dynamics), the characteristics of the sensor nodes, and the coverage of the phenomena. Figure 3-1 shows the characteristics of *WSNs* at different application sizes. The figure illustrates that the size of the network depends on the in-network collaboration. A single hop network with no collaboration is considered as a small scale network. A multi-hop (low/middle deployed) network with partial collaboration is considered as a middle scale network. A multi-hop (densely deployed) network with full collaboration is considered as a large scale network. The figure shows that as the network size increases from a small scale (of 20) to a

middle scale (of 100) and finally to a large scale of millions, the network configuration changes from being deterministic to self-organising. This changes the network intelligence from centralised to distributed intelligence which reduces the usage of resources and prolongs the network's lifetime. Moreover, as the network becomes larger, the processing and the data that are going to be gathered become more in-network.

The increase in the size of the network will reduce the quality of the nodes. This is because for large size networks the cost of manufacturing, deployment and maintenance of the nodes increases. In order to compensate for the cost of manufacturing of nodes, the nodes are manufactured at the lowest boundary technology. This increases the noise impact on nodes. Moreover, the reduction in the nodes' quality causes an increase in the packet loss. This is in addition to the losses that result from routing paths or impact of external factors, which increase with the increase in the network size. In dense deployed networks, the network uses the redundancy of measurements to improve the accuracy of the individual cheep nodes. This is not needed in low density deployed network because the nodes have more accurate sensors. Moreover, the figure shows that the information collection changes from raw data in small size network to aggregated data in large scale network in order to save power and reduce the usage of network constraints resources. Consequently, because of these different characteristics, different communication and application protocols have been used to ensure the utilization of these network characteristics and reduce the impact on network resources.

Because of the requirements of network diverse applications, network sizes, network design goals, and network characteristics, there are various metrics and methods that detect *WSN* performance. Zhao, in [14], argues that *WSN* performance evaluation can be divided into two main areas: performance measurement and performance monitoring. Performance measurement is carried out in *WSNs* in order to evaluate any design choices in real systems, quantitatively compare them, and understand any design constraints. This is particularly important at the design and evaluation stages. This type of

performance evaluation uses several metrics to compare the efficiency of protocols and network functionality. For example, the efficiency of network energy can be evaluated by measuring energy per correctly received bit, energy per reported event, delay and energy tradeoff, network lifetime, time to partition, time to loss of coverage, time to failure of first notification, and the probability of a node surviving for a given amount of time [1]. Besides these energy efficiency metrics, *QoS* can be used to measure low levels of characteristics in the network such as bandwidth, delay, jitter, packet loss, probability of event detection/reporting, event classification error, event detection delay, missing reporting, approximation accuracy, and tracking accuracy [1]. In addition to these metrics, some researchers have considered the cost of sensors and their deployment. All of the metrics mentioned above have to be evaluate under a clear set of assumptions about the energy consumption characteristics of a given node.

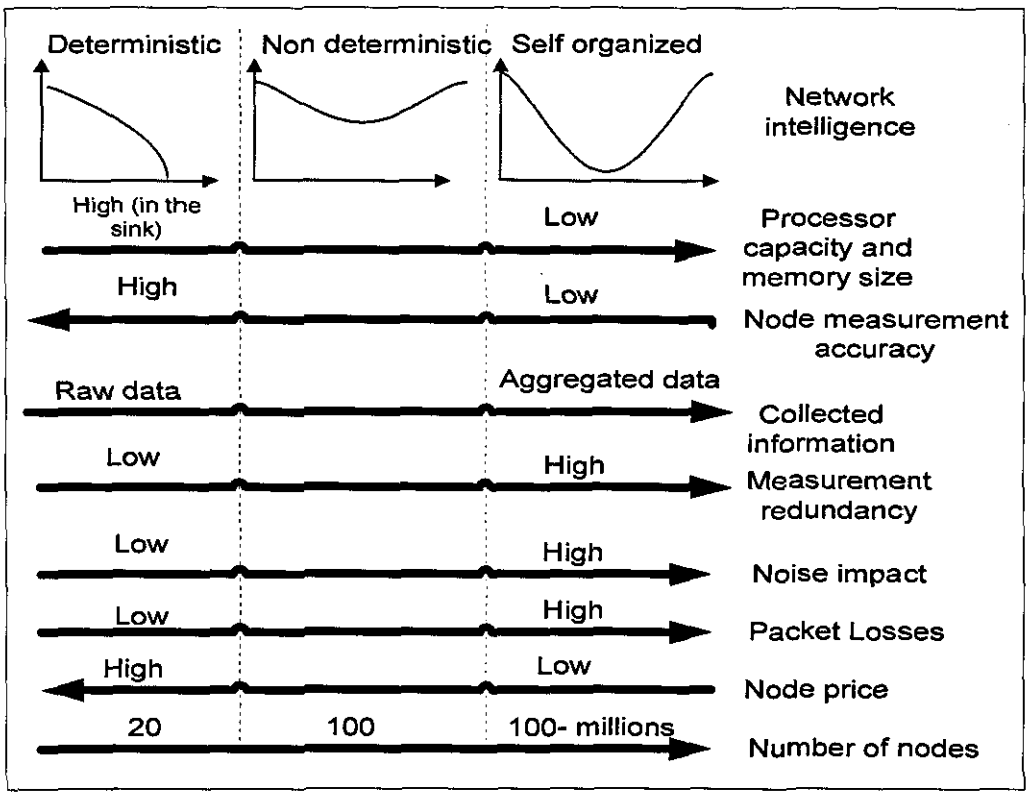

Figure 3-1. Wireless Sensor Network Characteristics for Different Implementation Sizes

On the other hand, *WSN* monitoring algorithms can be used to monitor the overall state of the sensor network, to provide early warning of system failure, and to examine the practical difficulties of precisely planning sensor field

deployments. Furthermore, they may be used to validate expected sensing functionality or for fine-tuning detected algorithms. The main metric used for this is energy consumption as in [14].

3.5 Identifying *WSN* Performance Metrics

In general, a *WSN* is programmed such that it completes the required monitoring within a given time range. This can be achieved by adjusting the following parameters; data collection period, sleeping time, in-network processing, and storage capacity in terms of certain specific delivery requirements, such as accuracy and delay, to control the network's lifetime, as discussed in [9]. Performance achievement in a *WSN* is a collaboration between different network components and, since this makes it unclear how to measure *WSN* performance, it makes the traditional techniques not suitable. Because of this, a new *WSN* performance tool and its tracked events can be only designed efficiently by identifying and understanding the following aspects of *WSN* functionality:

1) Defining the required interest of the network application, such as measurement accuracy, delay and any others.
2) Checking the characteristics of the protocols used their interactions and whether any global parameters are used for this interaction.
3) Checking the relation between the high and low network levels parameters, such as the application's accuracy tolerance to losses.
4) Choosing common parameters that can represent both levels and can indicate network degradation if this change within certain levels, such as distortion in the neighbourhood collected data.
5) Checking the level of performance accuracy required at the destination and the available resources which predict it.
6) Deciding how to present the measurements, the types of performance measurement, the rate of performance measurement, the location of measurement and the action to be taken after detection.

All the above-mentioned aspects ensure that the performance parameters that are chosen depend on common features available in the application in order to reduce the usage of network resources. They also identify the collaboration function that is used among network nodes and its resilience to different changes.

3.6 Locations of Performance Measurements

WSN protocols and algorithms are implemented either by a centralised or distributed mechanism; they are governed by the sensor nodes' capability and the application's requirements [60]. These implementations have been studied by many researchers, such as [8],[61], whose work found that there is a tradeoff between resource usage and the application's requirements where each implementation option has a cost in terms of its communication, processing, energy consumption and memory storage. Distributed methods involve storage which depends on the cost of code complexity, processing, energy and communication. The main advantages of such methods are the low level of communication required and the better scalability it provides as a result of the use of a distributed analysis. Centralised methods, on the other hand, do not need to consider usage of resources due to their availability (This is because these nodes are attached to computers that have relatively larger *CPU* and memory with respect to sensor nodes). But this method involves high level communication which can cause bottlenecks.

3.7 Proposed Performance Metrics and Events

As discussed in previous sections, each *WSN* application imposes specific requirements on node characteristics, node deployments, measurement precision, network protocols, and network traffic load. This makes it difficult to specify parameters so that there will be a general performance measurement suitable for all applications while using the confensional methods. However, to return to the definition of a *WSN* in Chapter 2, this showed that there are two main functions of *WSNs*: measuring phenomena and the transmission of these measurements. So, the health of a *WSN* can be judged based on the

accuracy of performing these two functions no matter which application or protocol is used.

The accuracy of these two factors, however, is affected by the accuracy of the measured data and the accuracy of the transmitted data. The accuracy of the measured data is a function of the sensor nodes' characteristics, the sensor transducers attached to them, node deployments and variations in the phenomena, as discussed in [8], [19]. Data transmitted accuracy, on the other hand, is a function of the protocols that control the nodes and make them follow certain collaborations while collecting and exchanging the measurements; this process is affected by the received packets and losses.

Since Wireless Sensor Networks are different from traditional networks in the functionality of nodes in the production, processing and exchange of information, these data not only represent measured phenomena, but also indicate the properties of the processed being conducted in-network. This is because they configure/organise the network [25], the effect of which depends on their location with respect to the phenomenon, the nodes' functionality, the accuracy of individual sensor measurements, network stability and sensor responses to external effects. Because of this, the data that are extracted from the environment as discrete samples of physical phenomena are a good indication of the network's health because they depend on node functionality, phenomenon changes and losses of packets between neighbour nodes. Comparing variations in node measurement with a neighbourhood reference will detect changes in the network's functionality and detect the level of accuracy of the data.

This thesis depends on above mentioned hypothesis to monitor the performance of *WSNs*. The algorithm proposed here extracts simple metrics available in all *WSN* applications (i.e. neighbour node identification, neighbour losses, and neighbour measurements). These metrics analyse events to measure the degree of change between neighbourhood node measurements, neighbour losses and the effect of these losses on the collected data and the functioning of the network protocols. They also analyse low network levels

(i.e. neighbour packet loss) to detect dead neighbour nodes, track changes in communications between nodes in the neighbourhood, and power efficiency. At high network levels (i.e. measurements), the events carry out an analysis to detect the malfunction of neighbourhood nodes, track changes in neighbourhood coverage, and check the efficiency of neighbourhood power consumption (i.e. in terms of measurement closeness and their reporting rate). Finally, these events relate the high and low level metrics in the network to track the accuracy of the collected data and their effect on network protocols. These analysed events, described above, rely on estimating the expected phenomenon measurement and evaluate the difference of individual neighbourhood node measurement to a specified threshold, depending on the application's goals. This will be discussed in Chapter 4.

3.8 Related Work and the Literature Review

Although the researchers' analysis of several real *WSN* deployments expected there to be an improvement of the deployed network's functionality of up to 51% if a real-time monitoring/measuring tool was used [62], the special characteristics of Wireless Sensor Networks, discussed in Section 3.3, lessen the functionality of these tools, reduce their efficiency and have a high level of impact on the lifetime of the monitored network. As a result, the researchers proposed several methods, such as a tradeoff between the algorithm's detection accuracy, its response time and its resource usage, to reduce the impact of such tools on monitor network. This was achieved by selecting the algorithm's parameters, controlling the required packet exchange, and controlling the level of complexity of the analysis.

The researchers selected the algorithm's parameters so that the required extraction resources had a low impact on the network's lifetime. This was done by using common parameters that are available at low or high network levels and relating them to network status. For example, at low network levels, they tracked packet losses between neighbour nodes and related them to network congestion, environmental effects, node battery depletion, and other

hardware problems; as discussed in [62]. While at high network levels, they tracked the changes of node measurements and relate them to sensor node software/hardware problems, as discussed in [8], [26]. Moreover, some of them were extracted at particular parameters that would track certain goals designed in the network, such as power consumption [40], node coverage and connectivity [21].

Also, they control the rate of exchange of the algorithm's parameters in order to reduce the impact of the monitoring tool on the network's lifetime. Some of them use techniques continuously to collect parameters and analyse them centrally, such as [14], [63]; others distribute these analyses to reduce the number of exchanged packets and the reply time, as discussed in [23], [63]. The third group use predicted models that exchange the packets if there is a large discrepancy between the actual and the predicted values, such as in [40], [63].

Finally, these researchers reduce the impact of the algorithm on the network's lifetime by controlling the complexity of the algorithm's analysis and the resources it uses. This is achieved by generating the residual of the monitored parameters in terms of physical or analytical redundancy, as discussed in [26], [64]. Physical redundancy generates an estimate of the actual value of a quantity based on the available redundant information; this is accomplished either by statistical methods (such as descriptive or inferential statistics), or by data fusion. The main advantage of this type of redundancy is that it is relatively easy to implement and provides a high degree of certainty; its reliability relies on the accuracy of the collected measurements. Analytical redundancy methods, on the other hand, provide values other than direct measurements from the parameters and variables of interest using a process model such as a Kalman filter, parity relations, Principal Component Analysis (*PCA*), and Artificial Neural Network (*ANN*). These methods are not easy to implement and they depend on the reliability of the process model.

All the above methods for extracting, exchanging and analysis were gathered together to ensure the reliability of data collected in the network by using four

main techniques: data cleaning, fault-tolerance, diagnosis, and performance measurement.

3.8.1 Data Cleaning Techniques and Rectifications

These techniques work at high network levels and consider the impact of deviated nodes on multi-node aggregation/fusion techniques, such as in [10],[12],[65]-[67], which propose several methods to isolate deviated data by tracking or predicting the correlation between neighbour nodes. Tracking methods use network redundancy and voting techniques to reduce uncertainty, reduce resource usage, and increase reliability in case of sensor error or failure. This allows the detection of environmental features using information from individual neighbour nodes, as in [68]. The main problem with these types of method is that they do not consider the impact of packet losses that deviate the estimated value from the correct value. Prediction methods, however, use various coding and storing scheme algorithms such as Linear Predictive Coding (*LPC*) and Discrete Kalmans Filter (*DKF*), as described in [55]. Most of this research used complex methods or models that need high levels of resource usage for detecting and predicting node measurements. In some cases, they need to have special nodes with large resources in order to function so they can be used with a single hop or small-sized network. In general both methods; i.e. tracking or predicting; rectify/clear deviated data after detecting them, but without checking the cause and the impact of these deviations on network functionality.

Our proposed algorithm uses a simple detection voting technique that compares readings from neighbour nodes with an estimated neighbourhood value (i.e. a median value).

3.8.2 Fault-Tolerance Techniques

Fault-tolerance and reliability performance techniques are important in embedded networks where physical access is difficult. This has been addressed at all levels in traditional networks including at circuit level, logical level, memory level, program level and system level. However, these

techniques have limited use within *WSNs* due to network resource constraints. Research such as [68]-[73] propose different fault-tolerant algorithms for the sensor networks with low resource consumption. Cheryan *et al.*, in [74], summarised these techniques by investigating, comparing and contrasting key algorithms for fault-tolerance in sensor networks. He described some of the existing versatile architectures for distributed sensor networks that explore fault-tolerant routing and sensor integration. In general, the proposed *WSN* fault-tolerant techniques detect faults occurring in: fusion and aggregation operations [70], network deployment and collaboration [71], coverage and connectivity [21], energy consumption [40], [41], energy event fault tolerance [74], calibration [26], reporting rate [9], detection of network phenomenon characteristics [68], and many others. These faults are detected using 0/1 decision predicates computed by individual sensors [23], [61], faulty sensor detection [7], [64], or event region and event boundary detection [75], [76].

Most of these techniques detect power change failure or node crash faults and either detect fault at high or low network levels, without relating the two to each other and without checking their impact on network functionality. The main drawback of these methods is the impact of unremoved faults on the network's functionality up to the time the fault is detected. This may cause a high level of resource usage and sudden disconnection of the routing path.
Our proposed algorithm detects deviations at both high and low network levels, and checks their impact on network functionality. Moreover, the suspected node, isolated within a monitor window, depends on the application's tolerance which reduces the impact of the deviated node on the network's functionality.

3.8.3 Diagnosis Techniques

These techniques use active or proactive monitoring to trace, visualise, simulate and debug historical network log files in real time and off-line, as discussed in [63], [77]. They detect faults after analysing changes. For example, Chessa *et al.*, in [78], considered the problem of identifying faulty nodes in *WSNs* by introducing the *WSNDiag* protocol which provides

diagnoses that take advantage of the shared nature of communications. Ruiz *et al.* in [79], used Management Architecture for Wireless Sensor Networks *(MANNA)* in order to evaluate a failure detection scheme, while Bokareva *et al.* [80] tracked correlation and [81]-[83] used confidence in evaluating the detection of faults. Nithya *et al.*, in [77], proposed a debugging system to debug low network level exchange in order to test network health. Her system draws correlations between seemingly unrelated, distributed events and produces graphs that highlight those correlations.

Unfortunately, most of these techniques are complex and send/receive test packets to confirm the detection of the fault. This causes higher resource usage, as well as increasing the traffic. This is because these tools assume a minimal cost associated with continuously transmitting debug information to centralised or distributed monitor nodes.

Our proposed algorithm detection depends on parameter changes in high and low network level metrics and their expected impact on network functionality. There is no need for send/receive test packets to confirm the detection of the fault since neighbours send *'No_Fault_Evidence'* messages if they disagree with a released warning packet. Moreover, with the proposed algorithm, a packet is released only when deviation from a group is detected or if a node calculation disagrees with a warning message.

3.8.4 Performance Techniques

These techniques are similar to diagnosis techniques but without iteration and send/receive test packets to confirm the detection of a fault. Although these techniques are useful when analysing the problems, in terms of characterising their nature and their impact on the network so that the network will not suffer a serious degradation in its functionality, there is little literature and research on systematic performance measurements and monitoring of Wireless Sensor Networks.

Zhao, in [11], [14], studied the effect of packet loss in *WSN* and presented the first complete work on measuring and monitoring wireless sensor

performance. He studied the effect of environmental conditions, traffic load, network dynamics, collaboration behaviour, and resource constraints on packet delivery performance using empirical experiments and simulations. His packet delivery performance study was based on the impact of these factors on network stability and in-network processing. He achieved this by checking the effect of losses on the reliability of node measurement in terms of the presence of wrong aggregate readings and the ability to distinguish these from correct ones. Although packet delivery is important in wireless communication and can predict performance by predicting in-network processing algorithm behaviour, it can give a wrong indication of the network's performance because of collaboration behaviour and measurement redundancy when the network's application is, to a certain degree, tolerant to it.

Zhao also proposed an energy map of sensor networks using the aggregation-based approach of *WSN* power consumption. His approach sends messages concerning energy level after every significant drop. Mini *et al.* and Song *et al.*, in [40], [41], proposed two different models of predicting node energy consumption and exchange their parameters with the sink to improve Zhao's algorithm. Anastasi *et al.*, in [84], measured nodes' performance by measuring the power consumption for different operationing conditions, checking the impact of weather conditions and neighbour interfaces with this. Although energy consumption is very important in *WSNs*, and all network levels are affected by it, as discussed in [24],[85], [86], several research studies, such as [87], showed in their analysis that there can be a sudden drop in node and network functionality which cannot be detected by voltage level. This causes instability in the network due to sudden route changes. It also increases energy consumption due to the usage of non-optimal routes and packet drops. This is used in our proposed algorithm and detects battery depletion by detecting both high losses and uncorrelated readings between neighbours. If these are different, it indicates with a strong probability that a node is faulty. Also, the algorithm adds a warning packet which is sent to the sink if the node stops its packet exchange due to power depletion. This is explained in Chapter 8.

3.9 Practical Implementation Techniques

For the practical implementation of Wireless Sensor Networks, Crossbow introduced MOTE-VIEW, Surge-Stats and History-View software for visualising network topology, network status and node functionality, network statistics, and for analysing network data, as shown in Figure 3-2 and Figure 3.3 [88],[89] . The problem with these programs is that they are limited to a maximum of 50 nodes with centralised monitoring. This needs a packet to be sent from each node, which makes the sink a bottleneck, and thus makes the software not suitable for large-scale aggregated data applications.

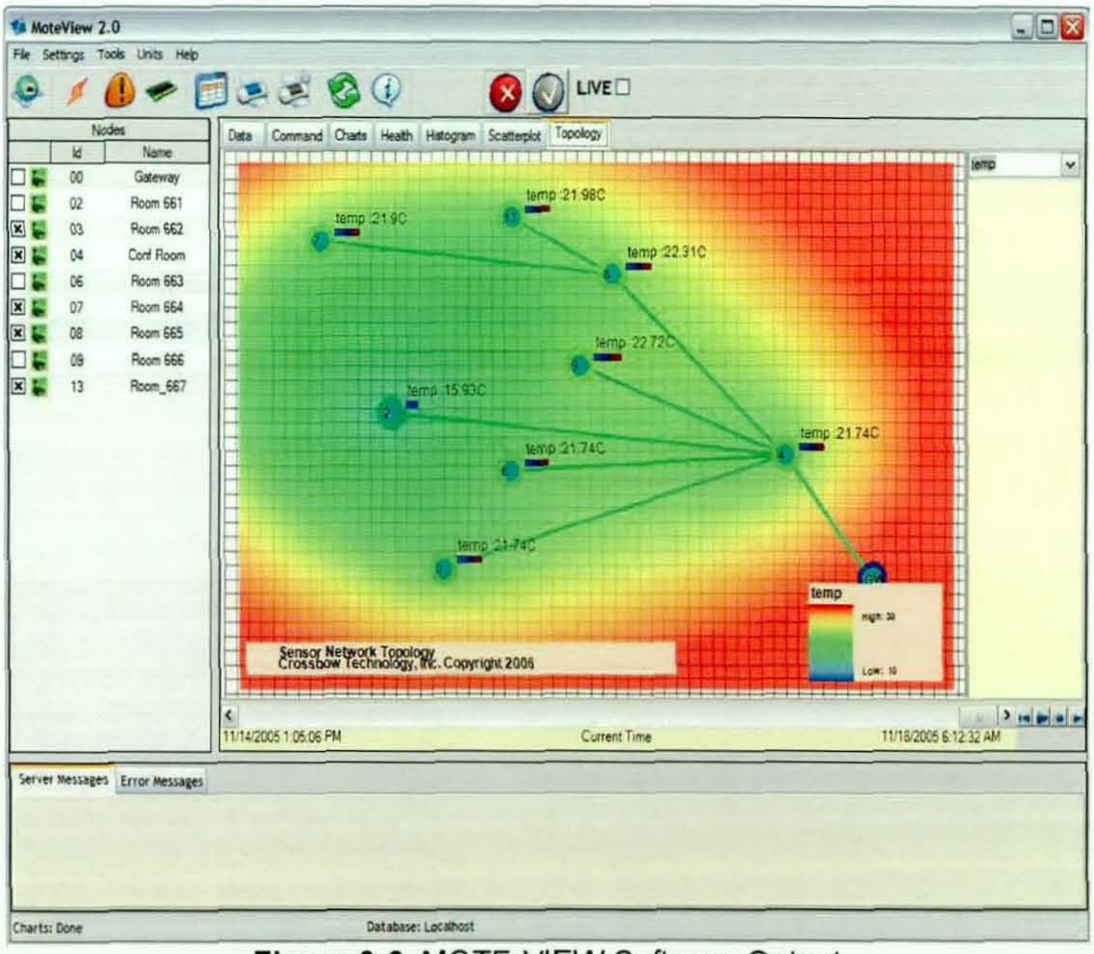

Figure 3-2. MOTE-VIEW Software Output

Our proposed algorithm is designed to work in a large-scale application (i.e. in a distributed manner) and its functionality is not limited to a set number of nodes. Moreover, it is a passive monitoring method and only sends warning messages to the sink whenever a network functionality degrade is detected.

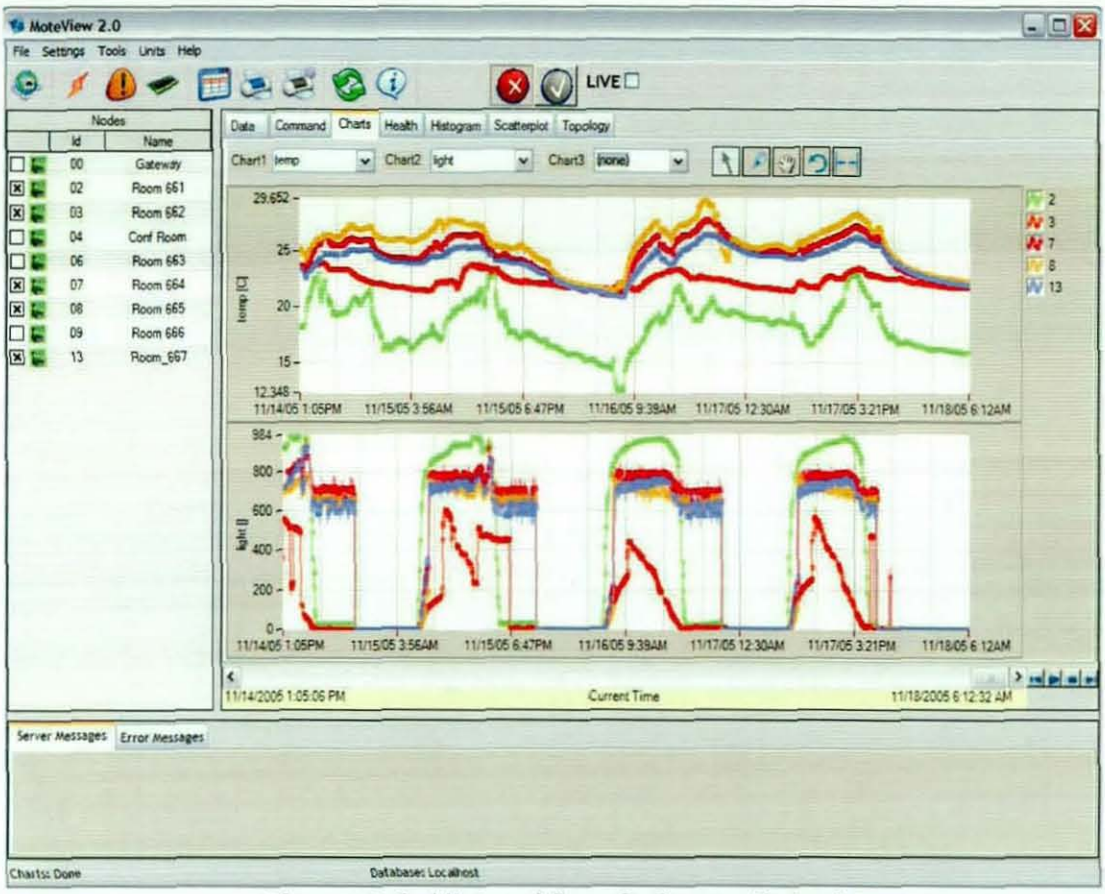

Figure 3-3. History View Software Output

I-Bean Evaluation Kits [90] are heterogeneous nodes that have different memory, processing capacity and power supply. These nodes do not participate in the network's functionality and have different frequencies for the communication amongst themselves and with the sink. In practical terms, it is not possible to distribute this type of node in most Wireless Sensor Network applications, such as large-scale applications in harsh environments. Our proposed algorithm can work with any node in the network and needs no particular size of memory or processing capacity.

Heinzelman *et al.*, in [91], proposed Middleware to measure node health. This uses a request-flooding communication between the sink and nodes that constructs a maintenance tree. When the sink detects a problem (i.e. when a path does not reply to the send request-flooding communication), it send and receive test packets to conform the detection of faults. This method consumes a good deal of energy by sending periodical request-flooding packets.

Moreover, it needs to build a global view of all the nodes, and their virtual connections, in the network.

Finally, Weizhong *et al.*, in [12], proposed a performance measurement algorithm for sensors located inside General Electric (*GE*) generators by examining the features of the frequency domain of the nodes. The problem with this approach is the algorithm's complexity which needs high processing power. As a result of this, it is only used in a centralised manner. Moreover, Weizhong did not consider the effect of packet losses on his algorithm. Our proposed algorithm can be implemented in any node because it does not need a sophisticated processing capacity or memory size; it also tolerates high packet losses, as discussed in Chapter 7.

3.10 Summary

This chapter has shown that large telecommunication networks performance techniques cannot be directly implemented in a *WSN* due to their high resource usage and the fact that the global parameters they use are unavailable. This has created a need for new performance techniques and metrics. The chapter argues that a unique *WSN* performance metric cannot be set up for all *WSN* applications due to the dependence of each network design on the application's characteristics since each has a different deployment, node characteristics and protocols.

The chapter also argues that the degree of change between neighbourhood measurement properties is one of the best ways of analysing events that gather both high and low network parameter levels and track the functionality of *WSNs*. This is due to nodes participating by both collecting and exchanging network information and by imposing any changes in measurement values. The detection of such degrees of change by the proposed algorithm, is carried out by comparing these measurements with a neighbourhood estimated value. Normal changes, that occur as a result of changes in phenomena or the environment, are with very small difference (i.e. correlated), and abnormal

changes, which arise as a result of deviations, are with difference exceed assigned goal level (i.e. uncorrelated).

Finally, the chapter reviews some of most important work that has been carried out in measuring and monitoring the functionality of Wireless Sensor Networks, as well as listing their limitations. It discusses how the proposed algorithm attempts to solve these drawbacks.

Chapter 4 Approaches to Monitoring Sensor Network Performance

4.1 Introduction

This chapter sets out to explain the architecture of the proposed algorithm, different modules, components, and the theories behind them. It begins by discussing the characteristics of the proposed algorithm, the metrics it uses, and the events it analyses. A discussion of the algorithm's different modules functions and detection level of confidence follow this.

4.2 Characteristics of the Algorithm and Assumptions

The Voting Median Base Algorithm for Approximate Performance Monitoring of Wireless Sensor Networks (*VMBA*) is a passive voting algorithm designed to detect deviations that affect the quality and quantity of data collected by the *WSN*. The algorithm adopts a fault-detection management approach in its functionality as a result of the importance of node operations in the network's organization and configuration. It extracts its metrics directly from the application by utilising the overhearing that exists in the neighbourhood, as a result of the wireless communication medium, in order to reduce the resource usage and packet exchanges. It passively tracks the health of neighbour nodes by analysing events that record similar changes between neighbour node measurements which arise because of the characteristics of the monitored phenomena and the near proximity between neighbour nodes. In addition, it uses node *RAM* as counters that increase or reset (depending on the algorithm's analysis) and uses a discriminative decision approach to reduce resource usage.

The main advantages of the proposed algorithm, besides its low usage of node resources, are its real-time distributed performance monitoring that reduces the dependency on centralised control, its high detection confidence due to the use of multi-parameters at high and low network levels, the passive test of its detection with neighbour nodes, and the low packet exchange that occurs while detecting degradations in network functionality. Moreover, the proposed algorithm is scalable to any large-sized sensor network due to its distributed function operation.

The *VMBA* algorithm has been designed to be as simple as possible in terms of its analysis and detection confidence in order for it to be implemented in the current sensor node platform that has constraints regarding memory, processing capacity and power supply.

Four assumptions were made while designing the proposed algorithm. The first assumption is the availability of measurement correlation between neighbour nodes. This assumption is valid because *WSNs* create partial functional redundancy among neighbour nodes to increase the robustness of a single point of failure, and for phenomenon coverage purposes. This correlation may not be available with small, single-hop *WSN* applications that send all readings to a sink. However, with these applications, health analysis can be carried out in the same way as with a traditional wired sensor where the sink has a level of high resources and can perform complex analysis. The second assumption is that the number of neighbour nodes should be more than one. This is required in middle-sized and large *WSNs* for network connectivity and coverage purposes. The third assumption is that these nodes have a fixed transmission range. (If the node transmission range is not fixed, the algorithm should calculate neighbour nodes). The final assumption is that nodes are correctly calibrated before the deployment begins.

4.3 Algorithm Metric and Event Detection

Communication between nodes in *WSNs* is greatly influenced by the application interests and is considered as a network protocol overhead, where nodes in the network receive data from neighbours, and process them. The network's configuration and functionality depend on these exchange packets and their contents [19]. This strategy was adopted in *WSNs* to ensure the robust operation of the network in a highly dynamic environment, as well as optimising the network's overall functionality. Unfortunately, this overhearing comes hand in hand with an increase in power consumption that may reach 60% in some cases [92]. This solved by power-saving techniques such as sleeping schedules and using a cluster structure; as discussed in [16], [94].

However, these power-saving techniques offer more challenges to network protocol operations, accuracy of collected data and network performance; therefore, they need to be considered in any protocol or algorithm design.

To overcome the challenges offered by network power-saving techniques and in order to reduce the resource usage of nodes, the *VMBA* algorithm depends in its operation on application flow process. This means that the proposed algorithm does not have a need for special timers to synchronise its functionality. In addition, the *VMBA* algorithm extracts its metrics from network application parameters (i.e. neighbour node measurements and neighbour node losses respectively, as listed in Table 4.1). This is to reduce the usage of node resources by reuse of the existence saved application parameters.

The proposed algorithm uses a simple analysis that calculates the neighborhood median, neighbour packet losses, and the degree of deviation of each reading from the calculated measurements median. Also, it calculates the weighted residual of each measurement from median, the degree of accuracy between neighbour measurements and monitoring node measurements, and the deviation of each neighbour received packet loss from the neighborhood's median loss. The algorithm analyses these events and tests them with thresholds in monitoring windows. The size of these windows and thresholds depends on the network design goals, the tolerance of the network protocols, and the required accuracy of the collected data. These analyses give the algorithm the possibility of detecting node malfunction, network functionality, and degrade in the network's connectivity/coverage as shown in Table 4.2.

Metric name	Metric Description
Neighbour node lists	List of neighbours of node identify by *ID*. The neighbour selected depends on the application.
Neighbour losses	Number of packet not received from neighbour nodes at the monitoring window.
Neighbour *measurements*	Phenomenon measurement value of neighbours.

Table 4-1. Metrics Collected from Application at Monitoring Window

Algorithm Detection Event	Description	Metrics used to *recognize event*
Node malfunction	Node measurements deviated from neighbourhood median threshold for monitor window.	Neighbour measurements; neighbour losses.
Coverage detection	Change in correlation between group of neighbour nodes.	Neighbour measurements.
Temporal coverage change	The algorithm detects and releases node malfunction several times within monitor window.	
Neighbourhood malfunction	High variation of neighbourhood median for monitor window.	Neighbour measurements.
Neighbourhood accuracy degradation of collected data	Degree of distortion of collected data accuracy due to loss of neighbour node.	Neighbour losses, neighbour measurements.
Aliveness	No packet received for monitor window.	Neighbour losses.
Connectivity degradation/ Connectivity instability	Neighbourhood median loss is more than 60%. The connectivity is unstable if the algorithm detects frequent disconnection of link between two nodes for three continuous monitored windows.	Neighbour loss.

Table 4-2. Detection Description of Algorithm Events and Metrics Used

The proposed algorithm tests the status of a node (i.e. if it is malfunctioning or normal) by checking the deviation of its measurement from the calculated neighbourhood median. This deviation is multiplied by a factor, depending on the number of nodes at that neighbourhood with the same measurement compared to the total number of neighborhood nodes received (or the level of deviation from the median). This is to test the impact of the environment and phenomenon change on changes in individual measurements. If the resultant deviation is larger than the threshold; i.e. its value depends on the required correlation between neighbour nodes at network or the expected value of the phenomenon at the end of the sensing range of the monitoring node; then this is detected as a node malfunction. If the algorithm detects more than one measurement that has deviated to the same degree from a neighbourhood estimated value, this is detected as a coverage problem. Similar deviations are considered as neighbourhood coverage problems because these nodes have detected a change in the measured phenomenon that other nodes in the neighbourhood did not detect, or these nodes are affected by a common environment that did not affect the others.

These detections are tested for monitoring windows to increase the confidence in the algorithm's detection and to differentiate between temporary and permanent changes where, if a node deviation stays constant for a long period, it is considered to be permanent. If, however, the algorithm detects a malfunction and clears it a number of times within a specified period, it is considered to be a temporary change. Figure 4-1 shows the absolute difference between the calculated neighbourhood median and node 13 measurements in the Intel *Lab* experiment data set [57]. The figure shows temporary and permanent changes that can be distinguished from their values and duration.

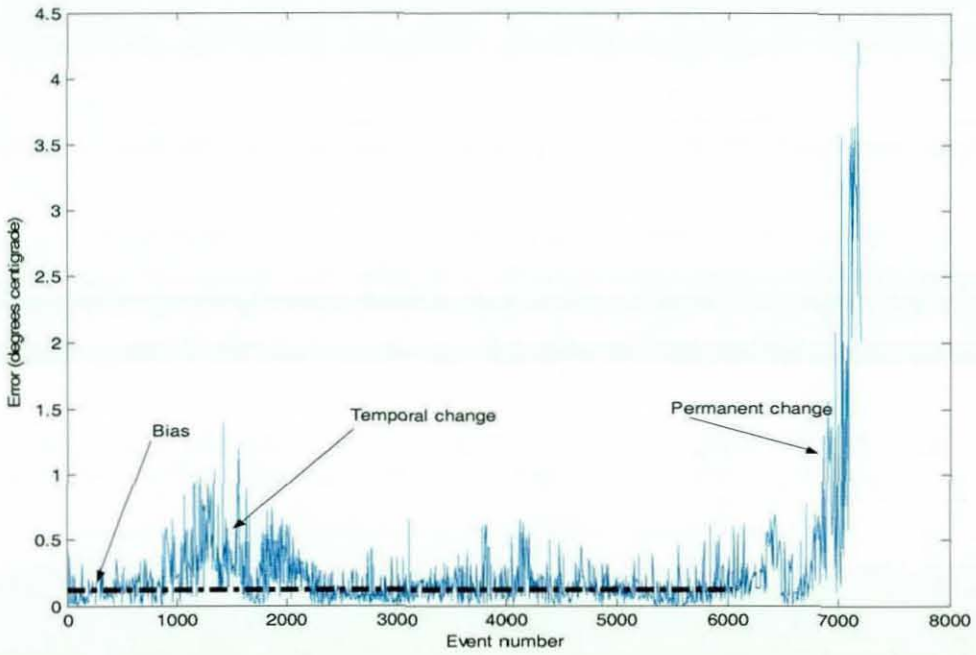

Figure 4-1. Difference between Calculated Neighborhood Median and Node 13 in Intel lab Data Set [57]

The algorithm also detects malfunctions in the neighbourhood by testing the effect of losses on the calculated neighbourhood median when a change is detected between two consecutive median calculations that are larger than the expected change in the phenomenon. The algorithm detects this change because it causes the communication and the application protocols to recalculate their tables frequently and to change data gathering points, data collection accuracy, and communication paths. Such frequent changes cause instability in the network; as discussed by Zhoa in [14]. If this instability is

detected for a specified period (that depends on network protocol tolerant with detected changes or the required detection confidence), the algorithm sends a message to indicate a neighbourhood problem. Figure 4-2 shows variations in the neighbourhood calculation median because of an increase in losses and an increase in the number of deviated faulty nodes in the Intel *Lab* data set for events above 60000. (This drawback concerning the median calculation was solved, as will be discussed later in this chapter but the variations are still detected by the algorithm to track neighbourhood instability and the effect of losses).

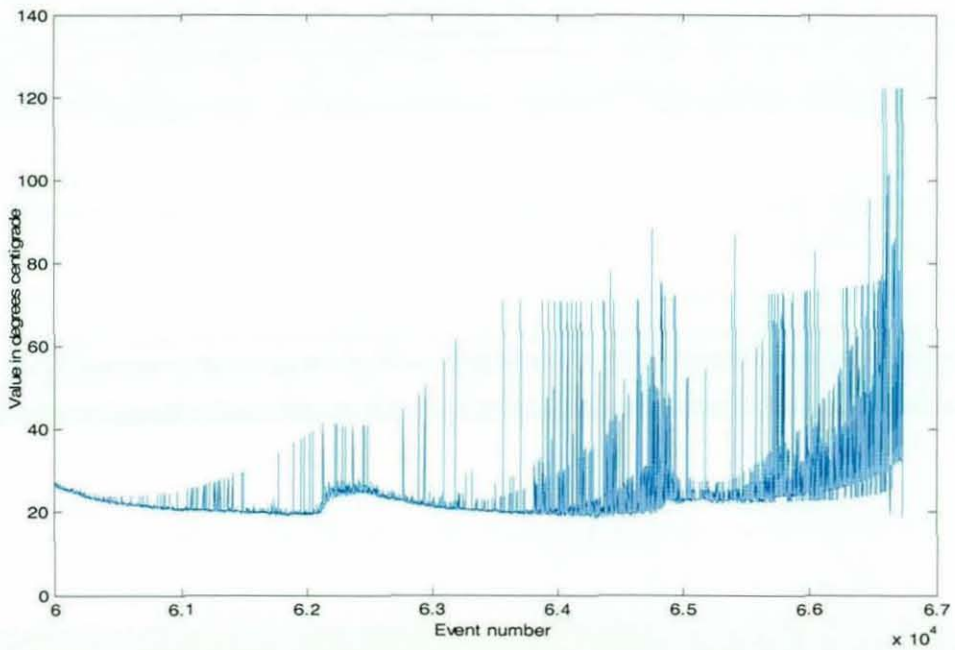

Figure 4-2. Neighborhood Calculated Median Value between Events 60000 and 67000 at Intel Lab Data Set [57]

The algorithm calculates the accuracy of data between the monitoring node and its neighbours. This is to detect the amount of losses that affect the accuracy of the neighbourhood's collected data. If these losses change the accuracy of the received data more than a threshold, the algorithm will detect data distortion.

The *VMBA* algorithm tests the efficiency of network power consumption and the communication status between neighbours by tracking the closeness of the measurements of neighbour nodes and the loss between nodes. If the

closeness in measurements is higher than the goals designed between neighbours in the network, and more than two nodes report the same measurement, the algorithm detects a situation of inefficient power consumption. On the other hand, if the existing closeness between neighbour measurements is less than the network's designed goal, and there are fewer than two nodes with a good link, (a good link is with one with less than 30%[1] loss [11], [14].), then the algorithm detects a communication problem.

The algorithm detects node aliveness by monitoring the packet released from a neighbour node for a specified monitoring time; this depends on the required response detection time and the detection confidence (as will be discussed later). If no packet is received from a neighbour node in a monitoring window, the node is suspected to be dead. If this detection alters several times within a specific number of monitoring windows (i.e. the monitoring node sends a suspected dead message then the node is heard again), the algorithm detects instability in the neighbour node connectivity. Furthermore, from these losses, the algorithm can measure inefficient power consumption by comparing the median of neighbourhood losses with a threshold. If it is higher than this threshold then the losses between nodes consume high power. (This depends on the probability of the required protocol stability and the significant oscillation of the tree structure of the particular operation). This may be due to either network congestion or environmental conditions.

Packet loss and deviation values are also utilised in the algorithm as parameters for controlling algorithm detection confidence; this will be discussed later in this chapter. For example, as can be seen from Figure 4-3 that illustrates Intel Lab data set node 5's [57] absolute difference from the neighbourhood calculated median; as a node becomes faulty, its measurement deviation value and losses increase from the calculated neighbourhood reference.

[1] This is because the experiments use the ratio of recalculating the count of aggregation tree (the recalculate interval) to the refresh time (periodical updates) as 4:1. When all the links face a loss of 30%, the probability that the link will reach the fourth period in the recalculating interval is 99%.

Because different *WSN* applications have different goals, the selection of these event analyses (as described above) depends on the application's interest, such as the detection confidence of the algorithm or resource usage. For example, when the algorithm was tested empirically on the *TinyOS 'Surge'* application (as will be discussed in Chapter 8), application losses were high and the accuracy between neighbours measurements were also high. Therefore, adding events to calculate data distortion did not succeed in adding any information to the algorithm's detection but only served to increase the complexity of the algorithm. Furthermore, power efficiency events release more warning packets that are not required and consume more power.

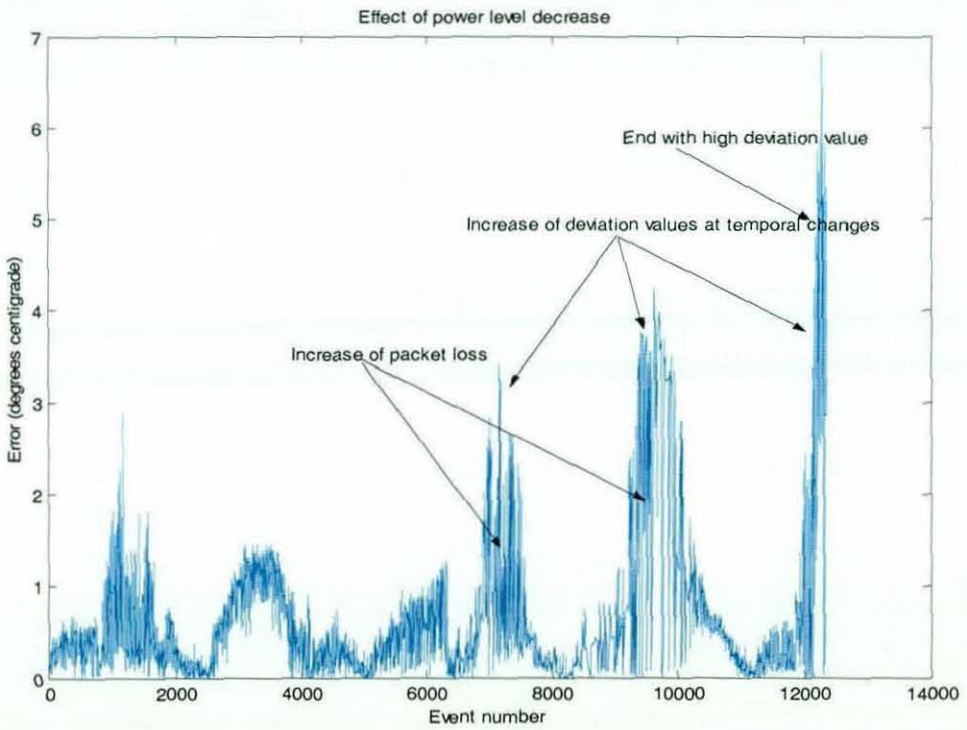

Figure 4-3. Difference between Calculated Neighbourhood Median and Node 5 in Intel Data Set [57]

4.4 The Algorithm Architecture and Mechanisms

The range of different researches which were discussed in Chapter 3, demonstrated that, in order to achieving the goal of the proposed algorithm, (i.e. detect deviations in the operation of nodes that will affect the quality and quantity of the data collected by the network and also have a low impact on the network's lifetime), the algorithm needs to be divided into four modules:

i.e. listening and filtering, data analysis and threshold tests, decision confidence control, and warning packet exchange. The different modules have been shown in Figure 4-4. Figure 4-5 shows the VMBA Algorithm Location in Sensor Application Flow Process. The process flowchart of the algorithm is shown in Appendix D.) This is because the main resource usages that affect the lifetime of the monitored network, and the detection confidence of the algorithm come from:

- The algorithm's collected parameters and the method used to extract them from the monitored network;
- The complexity of the analysis used;
- The uncertainty level of the algorithm's analysis;
- The algorithm's packet exchange.

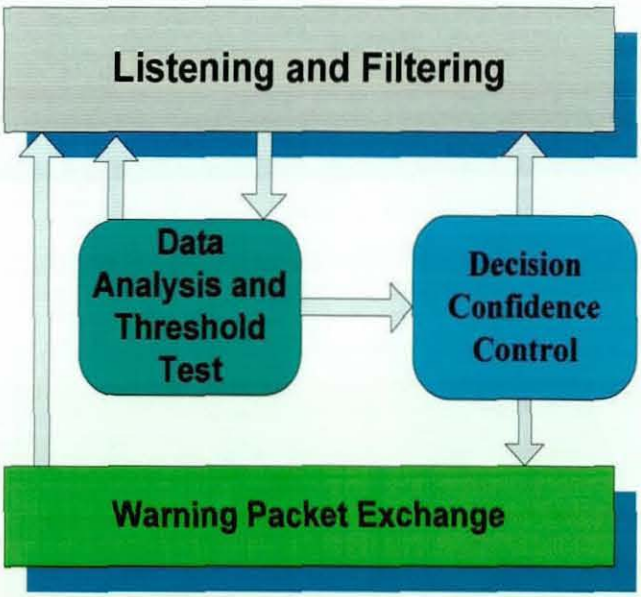

Figure 4-4. VMBA Different Modules

As a result, controlling individually each of the factors listed above ensures a reduction in the effect of the proposed algorithm on the network's lifetime and increases the algorithm's detection confidence.

4.4.1 Listening and Filtering Module

This module is responsible for filtering readings that are beyond the limits of the sensor node's physical characteristics after receiving them from

neighbours; it also constructs tables of neighbours' readings and calculates the median of neighbour node measurements received at monitoring time intervals. Moreover, it builds statistics concerning neighbour readings of loss at that monitoring window. This module is considered to be the most important module in the proposed algorithm because it is concerned with the construction of tables that the algorithm largely depends on in its analysis.

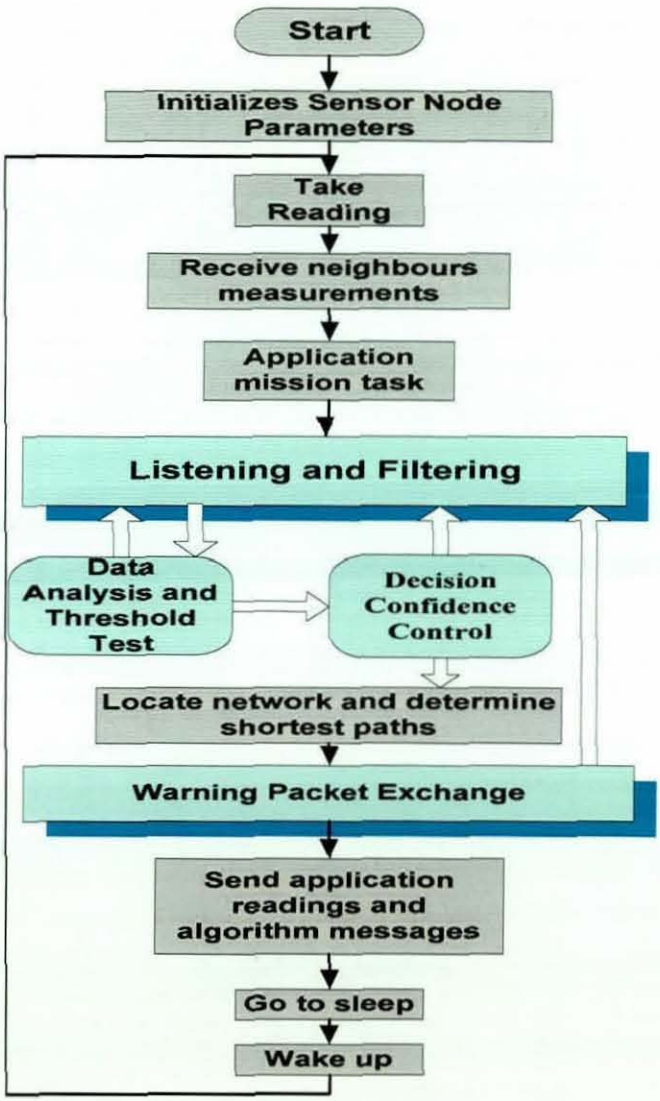

Figure 4-5. VMBA Algorithm Location in Sensor Application Flow Process

4.4.1.1 Filtering of Received Readings

The module starts its function when a sensor report is triggered, either by sensing an event, by a period timer expiring, or by a query from the sink. This

is followed by a waiting period that is used for receiving the measurements from the monitoring node neighbours, as shown in Figure 4-6. The notations used here are explained in Table 4-3.

1: Each S_i sense the phenomenon and wait for time T to receive N(S_i) readings
2: IF t > T THEN
3: For each unreceived x_j^i increment L_j^i;
4: IF $C_L > x_j^i > C_M$
5: Remove x_j^i from data set and increment D_j^i
6: Calculate med_i of the available S_i data set

Figure 4-6. Listening and Filtering Module Pseudo-code

- S_i: Monitoring node used in VBAM algorithm.
- T: waiting time that depends on the reporting rate, node location, and network synchronization time.
- t: Wating time counter.
- k: Number of neighbours.
- N(S_i): Set of S_i neighbour nodes; i.e. S_{i1}, S_{i2},..., S_{ik}.
- x_j^i: Measurement of node j received by monitoring node i.
- L_j^i: Loss counter at node j (by monitoring node i).
- C_L, C_M: Minimum and maximum limits of the sensor node that depends on its characteristics.
- D_j^i: Deviation detection counter of node j by monitoring node i.
- med_i: Neighbourhood sensed value median calculation made by monitoring node i.
- med_{i-1}: Previous neighbourhood median calculation.
- Δmed: Allowed change in the phenomenon characteristic that depends on the temporary and permanent precision of the application.
- M_i: Median deviation counter at monitoring node i.
- d_j: Deviation of node j from calculated median.
- R_i: Uncorrelated readings counter at each time interval.
- COV_j^i: Coverage problem counter of node j monitored by node i.
- N_i: Neighborhood malfunctions counter.
- Θ_M, Θ_C, Θ_d, Θ_w: Thresholds of median, coverage, distortion and monitoring window respectively whose values depend on the tolerance of the network protocol characteristics detected changes.
- ML_i: Median of L_j^i at the monitoring window size.

Table 4-3. The Notations Used in the Algorithm

This waiting period depends on the network's data delivery time, node functionality, and node location. The waiting time is controlled by the period between the two consecutive measurements and/or the sleeping node period, with a maximum time equalling the reporting period. (If the node sleeps immediately after reporting its measurement, the algorithm works at the neighbourhood cluster head.) Any unreceived readings within this assigned period are considered to be losses. (This waiting time can be adjusted depending on the application.) If more than one reading is received from a neighbour during this period (i.e. as a result of a higher reporting rate), the monitoring node will take the first reading at that time interval. This configuration of the neighbour readings' waiting time (i.e. the same as the application's reporting rate period), reduces the need for a special timer for the proposed algorithm; this reduces the usage of resources such as memory and processing time, and makes the algorithm function depending on the application's flow process.

After this, the module examines the validity of the received neighbour node measurements by filtering those readings beyond the range of the sensor's physical characteristics (simillar to what was discussed in [94]). If a reading is beyond the range, the algorithm will filter it before passing it to the algorithm for analysis and the node malfunction detection counter is incremented (filtering of these measurements prevents the monitoring node from wasting resources on obviously deviated faulty measurements). This high deviation in a measurement may occur while packets are exchanged due to the impact of external conditions, such as interference or hardware problems, on the node and on communication; these conditions were discussed in [27], [95]-[97]. Also, this may occur because of variable exchanges in the *TinyOS* code since Wen, in [13], found that *TinyOS* characteristics cause some node values to be altered accidentally when performing other tasks in between the processing; Wen called this process failure. During the empirical experiments, conducted on the Mote sensor network testbed at the High Speed Network research group at Loughborough University (*HSN*), this failure was found to be greater when losses increased or when source code complexity increased. (This will be discussed in Chapter 8.)

4.4.1.2 Construction of Neighbour Node Tables

This process is followed by the construction of neighbour node tables, depending on the application. This variation arises as a result of the contents of application packets where some applications send in guidance for packets, such as time stamp, synchronisation between nodes and sequence numbers, and others do not. This controls the algorithm's source code and the complexity of its analysis.

Constructing neighbour tables can be achieved by two methods. The first depends on the assumption that the application reporting rate is the maximum time that measurements from neighbours can tolerate. This time is used as a waiting period for neighbour measurements and unreceived readings during this period are considered as losses. This method is used in many applications such as cluster, aggregation and fusion types [98]. The main advantage of this method is its simplicity which is because less complex source code is required; this reduces processor and memory usage. Also, with this method, there will be no need for historical measurements since the collected data are online within the reporting period and the table is not affected by other network packet exchanges, such as sink query and routing packets. In this approach, the table adjusts itself at each time interval and in this way there is no chance for accumulated errors to occur. If neighbour measurements are delayed more than the reporting rate, due to congestion and high losses, there will be table misalignments and incorrect detection by the algorithm. This drawback can be solved by varying the waiting period.

This period variation depends on the maximum expected delay from the packet in the application. The expected delay in turn depends on the deployment, number of neighbours and communication protocols such as routing protocols, MAC protocols etc. In the empirical experiments the period variation is achieved by delaying the calculation of events from 5 to 10 intervals. The number of delayed event intervals depends on amount of memory available and the expected delay time of the packets.

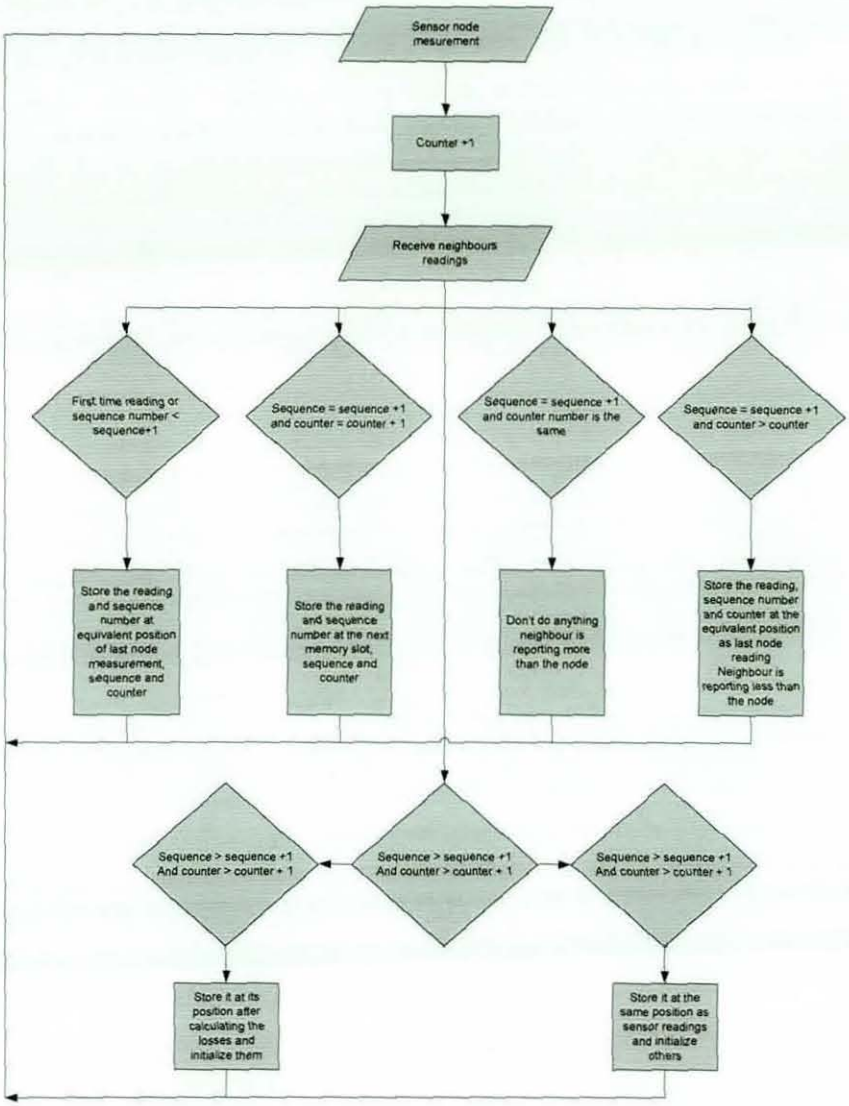

Figure 4-7. Table Construction Method

The second method uses more complex source code since it includes additional conditional statements and memory storage that require synchronised data to be collected from neighbours and then categorised, as shown in the flowchart in Figure 4-7. The problem with this method is that it needs a great deal of intelligence which increases the complexity of the source code (i.e. around 30-60% of the algorithm source code depends on the application). It also requires more *RAM* and *ROM* (as will be discussed later in this chapter). Another disadvantage of this method is that it can accumulate errors that cause neighbour reading misalignments. Moreover, it needs a lot of processing time. This causes a large amount of losses due to process failure and high percentages of packet repetition while sending some of the

packets. (This problem was faced when the algorithm was implemented in practice, as discussed in Chapter 8.)

Both methods construct their tables in *RAM* for quick retrieval while the data are analysed, which means that data are stored temporarily for the period while the algorithm analyses the calculations. Unfortunately, this produced the drawback of losing the received packets and the algorithm's analysis results if the node was switched off or initialised during its operation.

4.4.1.3 Median of Measurement Calculations of Neighbourhood Nodes

Nodes in large- and middle-sized *WSN* applications are deployed in a large number over a wide geographical area to increase the reliability of measurements, increase network connectivity/coverage, and to ensure uninterrupted and reliable operation. This makes *WSN* nodes geographically close to each other so they detect the same event at approximately the same time and have an overlapping sensing range. Moreover, if there are recognizable characteristics of a monitored phenomenon, such as a slow change characteristic, the similarity between the measurements of these adjacent nodes becomes higher.

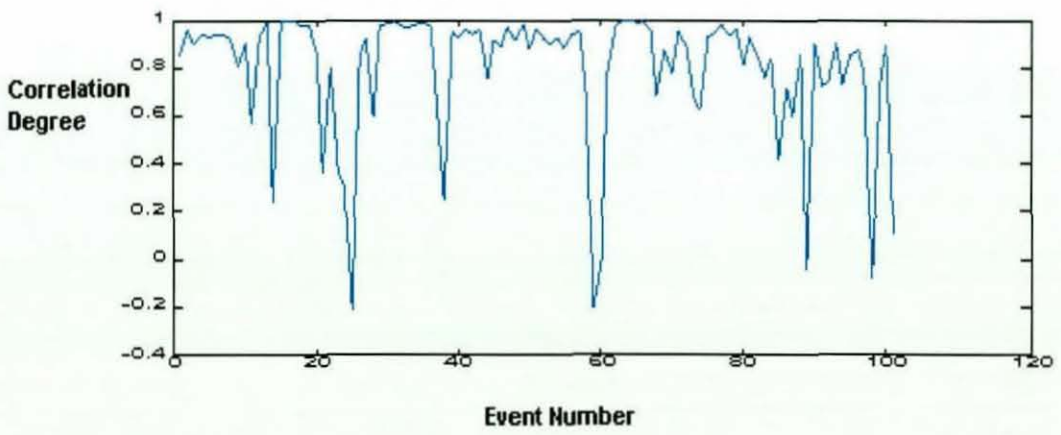

Figure 4-8. 100 Sample Size Correlation Windows between Two Neighbour Nodes Measuring Temperature

This similarity produces temporary and permanent correlation between neighbour measurements that can be tracked to measure the degree to which the two node measurements are similar or depend on each other, as

described in [21]. This similarity/dependency changes as the characteristics of the measured phenomenon change. The degree of change depends on the overlap of the sensed area, the degree of change in the phenomenon's characteristics, the physical characteristics of the sensor nodes, the monitoring window size, and the measurement techniques used to detect the correlations, as discussed in [26], [64], [101]. Figure 4-8 illustrates the degree of correlation between two sensor nodes measuring weather temperature one *Km* away from each other. The figure shows clearly that this degree of correlation is not constant and changes with time. This makes it difficult to track especially if resources are constrained and there are high percentages of loss. Due to this, many methods for measuring these related correlations are used in *WSNs*. These can be summarised from [93], [100] - [107] as:

- Limit checking of outliers' sensory data, such as Kalman filter and Fuzzy validation gates. These techniques compare the difference between the sensor readings and validated readings at a previous sample time, and the maximum change that is possible depending on the sensor's physical characteristics or the characteristics of the measured phenomenon.
- Redundant measurement values. These are divided into two main types: physical or analytical redundancy [26], [64] (as discussed in chapter 3).
- Probability approaches such as maximum a posterior (MPA), Neutral network and Kernel-based methods. These are used with no direct redundancy but are related to a group of sensors in the subsystem.
- Degree of correlation coefficient between different readings.

The differences between these methods stem from computational complicity, memory usage, data exchange, energy consumption, accuracy, and the location of the algorithm's implementation. Furthermore, in general, these techniques are divided into two main groups: prediction algorithms and tracking algorithms, (as discussed in [55]). The accuracy of correlation detection of the two groups depends on the amount of resources used for

storing and exchanging the algorithm parameters. Unfortunately, as well as the high level of resource usage, most of these methods are limited in their ability to detect sudden phenomenon/environmental changes; they may then consider them as faults since there is no basis of truth in the values of the measured phenomenon for their modules to depend on.

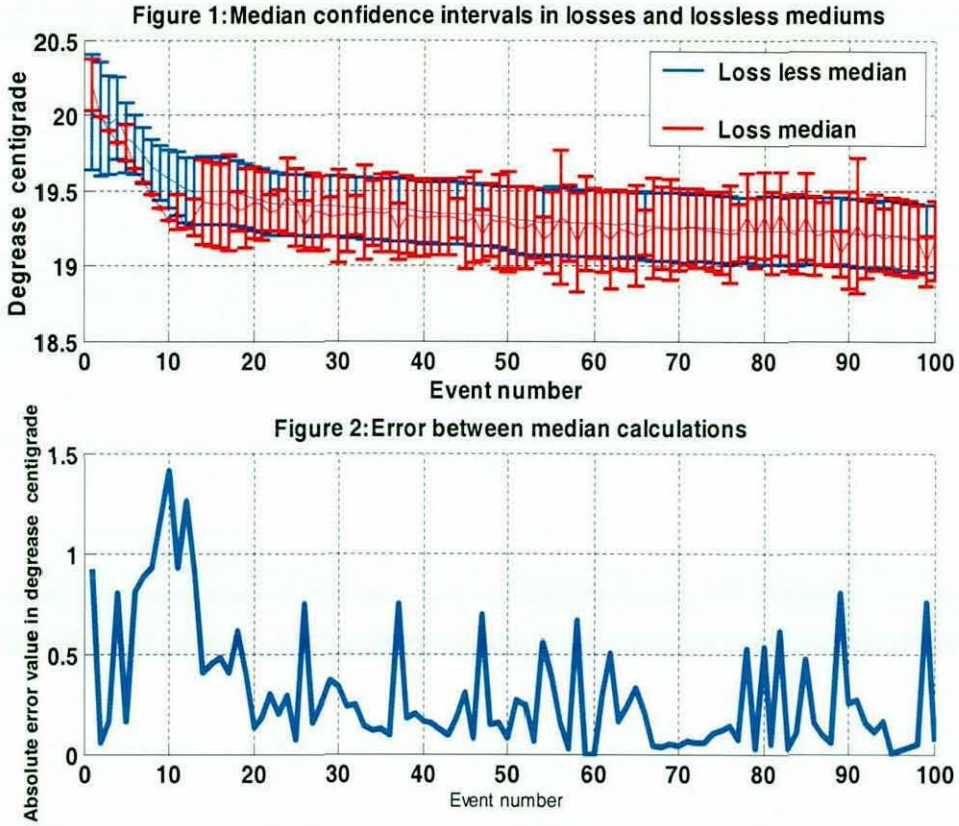

Figure 4-9. Loss Effect on Median Calculations

The proposed algorithm solved these drawbacks using neighbour redundancy measurements, accomplished through hardware redundancy, to deduce an estimated value of the phenomenon's measurement (i.e. the median). Median measurements were chosen because they are considered to be stationary statistics that minimise the influence of unbalanced, extreme data values [48], [108]. They also involve a simple level of calculation complexity. Wagner, in [109], found that the median calculation error level is only around $\frac{1.253\ \sigma}{\sqrt{n}}$ while its resilience is $k < \frac{n}{2} \approx \sqrt{1 + 0.101\ k^2}$; where n denotes the total number of readings, σ denotes the standard deviation of the readings, and k is the number of deviated readings. This low error level and the high resilience

increase the tolerance to loss of the median calculation. For example, Figure 4-9 illustrates variations in the confidence intervals of median calculations with zero and 65% packet losses for different events in the Intel Lab experiment data set [57]. The figure shows that the confidence of the median calculation changes along with loss, and the maximum average absolute difference between the two scenarios was only 1.4%. Moreover, the figure shows that the confidence of the calculated median depends on the number of readings used in the calculation, the percentage of losses, and the degree of closeness and/or conflict between data.

The two main problems of this approach are that there is insufficient information to isolate a fault resource in low density and low correlated networks (i.e. less than three nodes), and if the majority of readings deviate from a correct reading. These two conditions rarely occur because, when large Wireless Sensor Networks reach this stage, the network has a high probability of disconnection. Also, there is a very low probability of many neighbor nodes being faulty at the same time, with the same degree of deviation. (This probability increases in continuous reporting rate applications as a result of common functions [14].) On the other hand, the main advantages of the median calculation are its real-time use without the need of historical stored measurements, it is only slightly affected by packet losses, and its value does not drift to cause any change in a node unless a majority is affected.

Most researchers, such as [67], normalize the output of the analysis method in order to stabilise the results and reduce the effect of noise on the calculation; the proposed algorithm, however, uses this degree of variation to test the effect of noise on the status of nodes and the neighbourhood.

Furthermore, to reduce the median sort function analysis growth in resources usage from '$O(n^2)$' to '$n\log(n)$', a divide-and-conquer method was used, as explained in [52].

4.4.1.4 Packet Loss Statistics

Finally, this module calculates neighbour packet losses at monitoring time; as shown in Figure 4-6. The cause of such packet loss in *WSNs* can be divided into two main factors, depending on the place they occur [1], [11]. In the physical layer, this arises as the impact of variations in the characteristics of the receiver, the radio channel, and the sender circuitry or in battery levels (i.e. they are determined by the received signal strength). The second cause occurs in the *MAC* layer as a result of the communication generated in the network's topology which is controlled by a number of nodes that share that channel. Many researchers, such as Zhao *et al.* in [11], have experimented with packet losses in Wireless Sensor Network and have found that:

- The rate of packet losses change with time.
- The synchronisation loss is uncorrelated while the environment and network losses are correlated between the neighbour nodes.
- The difference between received and send loss of two nodes causes increase in communication losses (called *'asymmetric link'),* (Please note that radio links are normally assumed to be symmetric).
- The region around a node having a certain rate of packet loss is not circular but is shaped irregularly.

Although the experiment detected a high level of packet losses in *WSNs*, they showed that this affected not only the loss rate, but also that its effect on node protocol, computation and decision is important. For example, Zhao *et al.* in [11], empirically showed that, for a count aggregation tree, links of 30% were considered as good, while bad links were those above 80%. This is because the probability of stability in the aggregation tree with a 30% packet loss is 99% for recalculating the tree table (each recalculating period is after 4 update reports from nodes for their links). This means that the link has a high probability that it will not refresh (i.e. expire) before reaching the recalculate time. Refreshing nodes links before this period causes oscillation in the tree and errors in the aggregation calculation results.

As a result, each application has its own tolerance to these losses depending on the used protocols, number of neighbours measurements that are redundant and their characteristics. The huge number of packet losses in some *WSN* deployments must be taken into consideration when designing any algorithm or protocol. Their effect on the protocol must be fully calculated and tested, especially for the protocols that control the network, such as self-organize/self-configure protocols.

As a result of the characteristics of the losses described above, when the algorithm is assigned to calculate the losses of each link, it releases a large quantity of communication warning messages and connectivity. This has been solved by testing, together with the amount of losses, the percentage accuracy between readings and the monitoring node. If this accuracy was found to be lower than a threshold that affects the accuracy of the collected data, and there is only one link with a loss less than 30%, then the algorithm releases a communication problem warning. Moreover, the module assigns a counter for each neighbour node that increases when a packet is received. At the end of the monitoring window, losses are calculated for each neighbour node and the median of losses of the neighbourhood is then calculated. If no packet is received during this period, the dead node counter increments. If the median losses of the neighbourhood are above 60%, then a warning is released to indicate in-efficient power consumption problem.

This level of 60% was assigned because all our experiments depended on a *TinyOS* 'Surge' application, the functionality of which relies on the stability of the multi-hop routing (i.e. it depends on the tree construction while the topology changes). The stability of the relation between the child and the parent in this tree, as discussed in [14], was modelled as: $\emptyset(t) \approx 1 - p^{\frac{t}{T}}$ where $\Phi(t)$ is the probability that the parent node identifier will expire if a node does not receive any message from its current parent after a period of T, p is packet loss probability, and t is table updating time. The default relation between the two in the 'Surge' application is $t=5T$. Each node in this application sends, by default, information on its neighbour's link status every

10 seconds; this is recomputed after 50 seconds (i.e. routing table recalculation time). If it is assumed that the probability of parent/child refresh is not less than 90% (so that there will be no oscillation in the ad hoc tree that will cause communication failure), then the maximum probability of loss will be approximately 60%. This ensures that a link of 90% is maintained for the five relative states (i.e. 5 reporting time) without refreshing, even if the link suffers 60% packet losses.

4.4.2 Data Analysis Module and Threshold Tests

Nodes in *WSNs* are not only important for data collection and network communication, but are also considered to be a source of degradation in the network's functionality due to their importance in the organisation and configuration of the network. These degradations occur as a result of the degree of change between neighbours' measurements. These not only depend on the monitored change in the phenomenon, but the effect of the surrounding environment on the sensor responds; as discussed in [94]. In general, sensor node components are prone to faults because of the cost and quality of their components, the manufacturing process they pass through, the complexity of the applications' conditions, the direct interactions of the nodes with a harsh and hostile environment, and nodes' limited resources [8], [13]. These factors lead to different representations of the measured phenomenon at each node which changes its similarity to its neighbours. Moreover, it leads to a change in the network's functionality in terms of data gathering points, the accuracy of the collected data, and resource usage on both a temporary and a permanent basis [25].

The described changes occur either in a temporary or a permanent basis. Temporary changes occur either intermittently or transiently as a result of temporary external or internal conditions, such as changes in environmental conditions, interference and software bugs. The network returns to normal after the condition goes away. On the other hand, permanent changes, such as permanent hardware faults, for example, are continuous and stable in time. Such changes are caused mainly by hardware faults, reductions in

operational power levels, or the continuous impact of internal or external effects. These changes continue until the fault is rectified or removed.

The second module of the proposed algorithm adopts a voting method in its threshold scheme to distinguish between changes that occur due to variations in phenomena, environmental effects, or the previously described malfunctions in node resources. A voting technique was adopted because it is simple and does not require a probability distribution function for tracking changes. The assumption made in this module is that faults are likely have an uncorrelated degree/time and changes in the measured phenomenon, and in environmental conditions are spatially or temporarily correlated in degree/time between neighbours.

1: IF $|med_i - med_{i-1}| > \Delta med$
   Increment M_i and let $med_i = med_{i-1}$
2: $d_j = |med_i - x_j^i|$
3: IF $d_j > \Theta c$ and $|x_i^i - x_j^i| < \Theta c$
4: Increment COV_j^i
5: ELSE increment R_i
6: IF $\frac{R_i}{k} > 40\%$
7: Increment N_i
8: IF $(1 - \frac{R_i}{k}) * d_j > \Theta c$
9: Increment D_j^i

Figure 4-10. Data Analysis and Threshold Test Module Pseudo-code

The second algorithm module starts its operation after receiving the calculated median value and time instant measurements from module 1. It calculates the difference between the last stored calculated neighbourhood median and the new received calculated median in order to validate the accuracy of the newly calculated median value. If the difference is larger than the threshold Δmed, this represents the maximum expected change in the measured phenomenon within the period of the monitoring time. The module

then increments the accuracy degradation counter in the neighbourhood (This counter counts the number of times that the median calculation of the neighbourhood deviated from the expected change between two consecutive calculations. This deviation comes as a result of the increase in the number of deviated/faulty nodes at the neighbourhood and packet loss per event); and replaces the new calculated median with the stored median value, as shown in Figure 4-10. This is carried out to reduce the effect of neighbour packet losses on the median calculation, especially when the number of deviated measurements increases.

After that, the measurements are tested to detect the possibility of instantaneous degradation in functionality in both the node and the network. This is achieved by evaluating the contents of the module 1 tables and assigning them to dynamic or static thresholds after subtracting neighbourhood node measurements from the median; i.e. carrying out a residual calculation. If the subtraction is higher than the threshold value then it will be multiplied by the one minus the percentage of nodes having the same deviation value in the neighbourhood. This is done to check the weighted residual of the detected deviation and to test its effect on both the collected data and the network's functionality. (This weighted residual is adopted in the algorithm to solve the lack of necessary performance guarantees in terms of error rates in the voting scheme used). If the weighted residual is higher than a certain percentage, and if only one measurement shows this deviation, it is considered to be a faulty deviation and a flag is set at a detected time interval. If there is more than one deviated node in the neighbourhood, it is considered to be coverage problem; as shown in Figure 4-10.

4.4.2.1 Sensor Measurement Threshold Setting

A Threshold can be defined as the limit that must be exceeded to begin producing a given effect. Its reliability depends on the level of uncertainty, which tries to quantify the error. This uncertainty is calculated by using two methods: univariate and multivariate [64].

The univariate method explores each variable in data set separately. It looks at the range of values and the central tendency of the value. For this method the distribution of measurement error and/or knowledge about the process are used to detect significant deviation and estimate the uncertainty. The method of analysis for error detection depends on the number of sensor nodes, sensor measurements, and the correlation among the sensor measurements. Wen, for example, in [13], achieved a sensor characterisation model by transforming the analogue reaction which resulted from variations in physical phenomena. He created a physical map from the digital readings by repeating the measurements for each node in the network and for different measurements such as temperature, humidity and light; this took a great deal of time and effort.

The Multivariate method describes a collection of procedures that involve observation and analysis of more than one statistical variable at a time. It estimates the uncertainty based on the variables linked to the measurement like process noise, sensor drift, and sensor malfunction. In this method every feature is categorized by characteristic such as range and mean from which measurement features can be evaluated. The main disadvantage of these methods is that they give a large variation of the model standard deviation (as a result of the fault). In addition the high faulty data and high losses move the model range. Example of this method is the work of Elnahrawy et *al*, who in [8] used dynamic modules; these require more memory; historical measurements storage and complex analysis. This is difficult and uses a great deal of resources when used at node level and therefore such method is not feasible for use in large-scale networks.

Our proposed algorithm has followed a straightforward approach in calculating deviations in sensor functionality. Its analysis assumes that true measurements of a phenomenon's characteristics, following a Gaussian pdf, centre around calculating the median of neighbourhood readings. Any variance is controlled by the correlation expected at the end of the sensing range of a node, plus the sensor node's measuring accuracy (i.e. calculated from the phenomenon's power dissipation model similar to what discussed in

[110] or network correlation and collected data distortion as discussed in [21]), as shown in Figure 4-11. As a result the algorithm simply considers the change between measurements for efficient code operation and minimum resources usage (I.e. the algorithm functionality tracks the expected phenomenon value at the end of the node receiving range). However, any statistical distribution (with predictable change) could be used. The selection of any of these predictable models will not effect the algorithm's functionality.

To formalise this, let 'med_i' denote the calculated neighbourhood measurement median at a time instant ' i', let 'ΔX' denote the expected change of measurement at the end of the sensing range of the monitoring node, then, if ε_0 is the accuracy of the sensor, the expected value of the measurement of the sensor node n T_n is:

$$med_i - \varepsilon_0 - \Delta X \leq T_n \prec med_i + \varepsilon_0 + \Delta X \qquad (4.1)$$

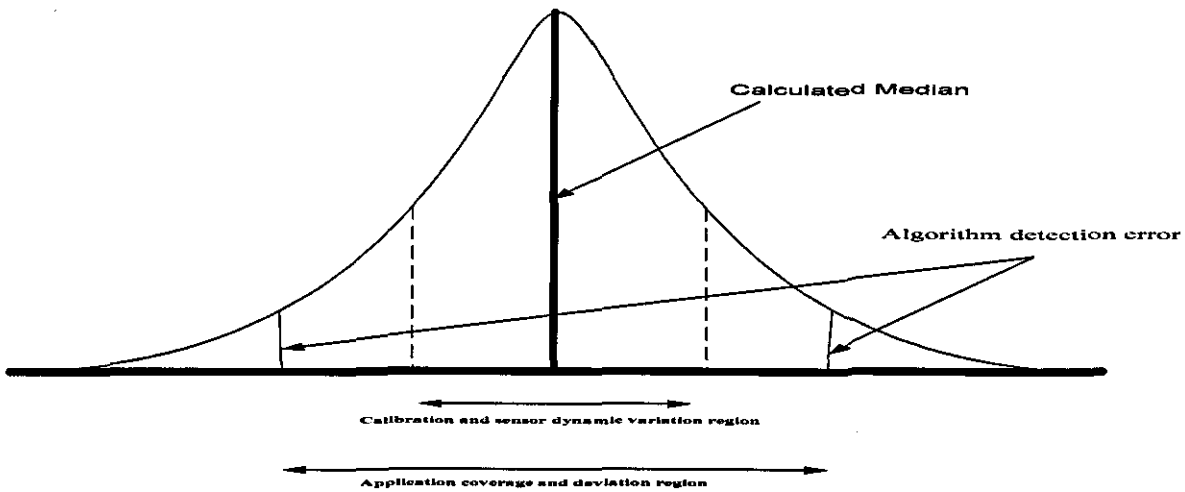

Figure 4-11. Probability of Calculated Median and the Variance around it

This assumption is based on the fact that the *WSN* design goals are achieved through a tradeoff between different aspects of the network's performance, such as power consumption and measurement accuracy. These tradeoffs are controlled by the characteristics of the network's nodes, the deployment of nodes in the network, network connectivity, network coverage and the complexity of the network's protocol. Any change or error in these control factors changes the functionality of the network. This deviates the network's

function from its targeted performance and can be detected from the node measurements in the network. (This is because the change in these tradeoffs changes the similarity of neighbour node measurements).

Because of this, the event analysis of the proposed algorithm considers any sensor measurement that is not in the threshold interval to be deviated to a degree equal to the ratio of the distance from the neighbourhood median value to the median value. This is because any external impact will affect all neighbours at the same time but to a different degree depending on their location from the nodes and the position of the nodes from each other. If the impact were different from either of these, there is a change in the response to this effect from other nodes or a change in the nodes' coverage of the measured phenomenon.

The only thing that should be taken into consideration while calculating the threshold value is to make sure that the interval is within an acceptable level of reliability, which depends on the network user's requirements and network protocol-tolerant. This requires some knowledge of the characteristics of the monitored phenomenon, the specification of the sensor node used, network protocols and the relation between neighbour nodes. This information is taken from the application deployment specification that the user needs know before network deployment, as discussed in [111].

Another way of setting a deviation threshold is by allowing the node to set it dynamically, depending on its deployment and changes in the phenomenon's characteristics in the field. (This will be discussed in Chapter 8.) An advantage of this method is that it takes into account the percentage of correlation at the actual deployment. Its main disadvantage, however, is the time required until the threshold value is set; this depends on the reporting rate, the number of neighbour nodes, and the percentage of losses, as will be discussed in Chapter 8.

Both dynamic and static threshold settings will change dynamically, along with changes in the collected neighbour measurements, as shown in Figure 4-12.

This figure shows the change in the algorithm's threshold limits along with the change of measurement in the neighbourhood for the Intel Lab experiment data set [57].

4.4.2.2 Relating Several Calculated Parameters to Detect Events

Some event detection in *VMBA* is carried out by relating several parameters to each other at the end of a specified monitoring window, as shown in Table 4-2. This checks the impact of losses on recovering the characteristics of the measured phenomenon by measuring the accuracy between two neighbour readings using Formula (4-2). This calculates the quality of the data's accuracy with the formula used in [112].

$$Accuracy = \left(1 - \left|\left(\frac{node\ \ reading\ \ - neighbor\ \ reading}{node\ \ reading}\right)\right|\right) * 100 \qquad (4\text{-}2)$$

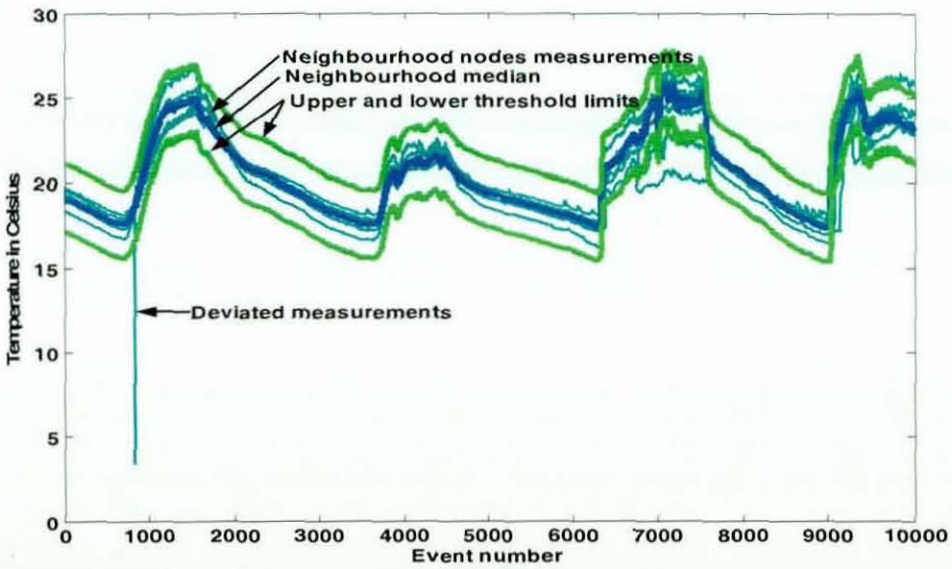

Figure 4-12. Dynamic Limit Variations with Neighbour Node Readings and Median Calculations

Another way of doing this is simply by calculating the number of nodes that have deviated from the neighbourhood median. If the number is higher than or equal to 40%, this definitely affects any self-configured protocol and a warning message is released by the algorithm (Else, a significant percentage may calculate, depending on the sensitivity of the protocol). The reason for 40% is that median analysis will deviate from the correct value if 50% of its

neighbours deviate. In order to be on the safe side we make it 40%. However this value can be adjusted depending on the application and used protocols.

The calculation of such threshold, for example, in aggregation techniques (i.e. max., min., mean, or sum) depends on the resilience of the individual method to the number of deviated nodes and their level of deviation, as discussed in [109]. This is because an individual deviation from the norm has an effect on the method's resilience and its calculated value. However, in methods such as fusion protocols such as [66], this can be done, depending on the percentage of deviated nodes in the neighbourhood, because of the high resilience of the method to individual change. (This approach has been used in our empirical experiments.)

```
1:  Calculate ML_i
2:  IF  ML_i > 60%
3:    Send to module 4 a request to send an inefficient power
      consumption warning message
4:      IF M_i > Θ_M
5:        Send to module 4 a request to send a neighbourhood
          malfunction due to losses warning message
6:            IF COV_j^i > Θ_C
7:              Send to module 4 a request to send to detecting node j
                a coverage problem message
8:                    IF distortion > Θ_d & median of L_j^i > 60%
9:                      Send to module 4 a request to send a degrade
                        detection in network functionality message
10:                       IF D_j^i >  Θ_w
11:                         Send to module 4 a request to send a
                            detection of node j malfunction message
```

Figure 4-13. Decision confidence control module Pseudo-code

4.4.3 Decision Confidence Control Module

The third module of the algorithm is concerned with a decision-making framework to detect changes in the health of neighbour nodes and in network functionality in assigned monitor windows; as shown in Figure 4-13. These are set depending on the characteristics of the network's application, its

design goals and the required response time. Reaching the predefined window threshold releases a request to module four to send a warning message. This reports a suspect node, the type of fault, the number of times the algorithm detects it, and the effect of the detection on the collected data and communications of the neighbourhood; as will be discussed in Chapter 8.

4.4.3.1 Monitoring Window

In data stream applications, such as *WSNs*, data are continuous, ordered and occur in real time so the required information can only be extracted from the collected data by a window over a period of time and then analysing the data within that period; as discussed in [9], [21]. Monitoring window functionality depends on the monitoring time period that has been set to detect the necessary deviation. This should be small enough and have enough data to maintain a certain level of accuracy, reduce the usage of resources, and reduce the fault detection time. Its size should be suitable for each application, in terms of the length of time and the number of samples, so that it can detect the deviation at a time when that deviation will not affect the functionality of the network and so that the detection is within the application's confidence level for data accuracy.

4.4.3.2 Types of Monitoring Window

When the algorithm was tested during the empirical experiments, some positive false warning messages were released due to the effect of losses, the high dynamics of the network and process failure. As a result, three windows were designed to reduce this effect: two for network functionality monitoring (i.e. large and small monitoring windows), and one for message exchange monitoring between neighbours. This was done to increase the confidence of the algorithm's detection and to detect both temporary and permanent deviations. However, this came with an increase in the algorithm's response time and also added more memory.

The design of the large window size depends on the level of confidence required, the application's reporting rate, the application's time tolerance to

deviations, and the degree of degradation in the accuracy of data readings that the application can tolerate (i.e. those calculated from the protocols used). This large window was divided into small sub-windows that were designed to detect the events independently at a time where this detection would have no historical impact. This was done to ensure detection and to reduce the number of deviation readings from old windows that might affect detection decisions. Also, these small windows indicate any temporary changes that the big window might fail to detect.

These two windows can be either static or dynamic; the proposed algorithm depends on static windows, however. This is because the proposed algorithm needs to detect changes if there is a continuous/frequent weight residual from other neighbours that affect the quality of data collected by the network. Sapon *et al.*, in [93], proposed the use of a dynamic window size to allow small variations to made in bigger windows. By decreasing the window size, the probability of un-correlation increases due to intrusion. In the proposed algorithm, this was achieved by using two sizes of monitoring window.

The efficiency of the algorithm's detection was checked and tested by using three different types of window, all with different characteristics such as memory usage and response time.

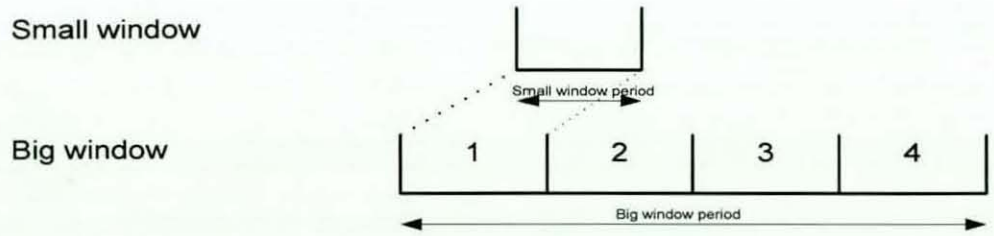

Figure 4-14. Periodic Detection Windows

4.4.3.2.1 Periodic Window

This type of monitoring window evaluates instantaneous algorithm analysis detection in two windows (i.e. small and large, as shown in Figure 4-14). Each big window consists of four small windows so that each is a quarter of the whole. At the end of the large window, the counters are reset and start to

recalculate changes without considering the effect of changes that stem from old data deviations. This is shown in the process flowchart in Figure 4-15.

The main problem with this type of window is that it depends on a delay in detection on the fault occurrence time with a maximum value equal to the sum of the big window's threshold time and the big window. For example, if the small window is set to two minutes, the big window is set to eight minutes, the network reporting rate is one second, the threshold of the small window is 80%, and the threshold of the big window is three small windows, then the maximum detection delay is 840[2] seconds, while the minimum is 480 seconds with a difference of 360 seconds. This delay increases linearly as the size of the small window increases, as shown in Figure 4-16. This figure illustrates the application's required detection time with a one-second reporting rate and the threshold of the big window set at a trigger of three small windows and at 80% of the small window size.

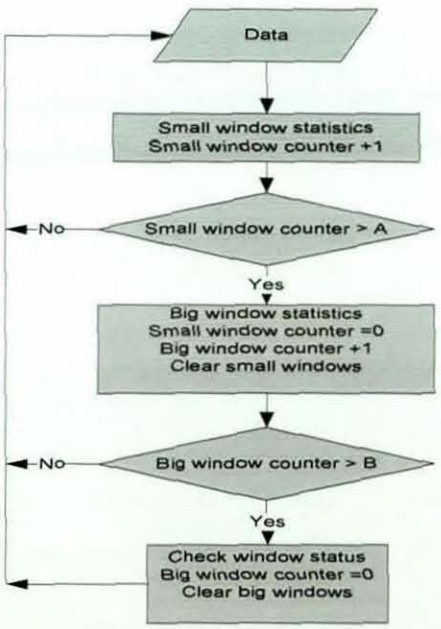

Figure 4-15. Periodic Window Functionality Flowcharts

4.4.3.2.2 Sliding Window

This type of window reduces the delay in the detection time, faced when using a periodic window, by keeping the historical weight for old detections. This reduces the detection response time below that for a periodic window, as

[2]That is 480+360

shown in Figure 4-16. The window function depends on replacing the oldest window slot by current monitoring values and recalculating the big window's parameters. (This is shown in the processor flowchart and block diagram in Figures 4-17 and 4-18 respectively.) The main problem with this window is the memory used by the counters of the historical small windows.

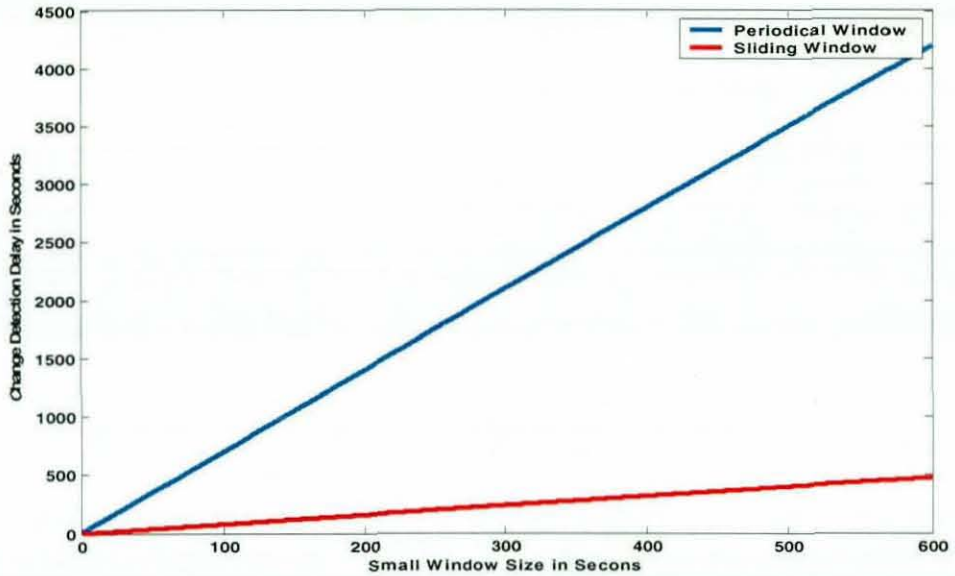

Figure 4-16. Relationship between Fault Delay Detection and Small Window Size with 1 Second Reporting, 4 Small Windows for and Big Window, 80% Small Window Threshold and 3 Small Windows Threshold for Big Window

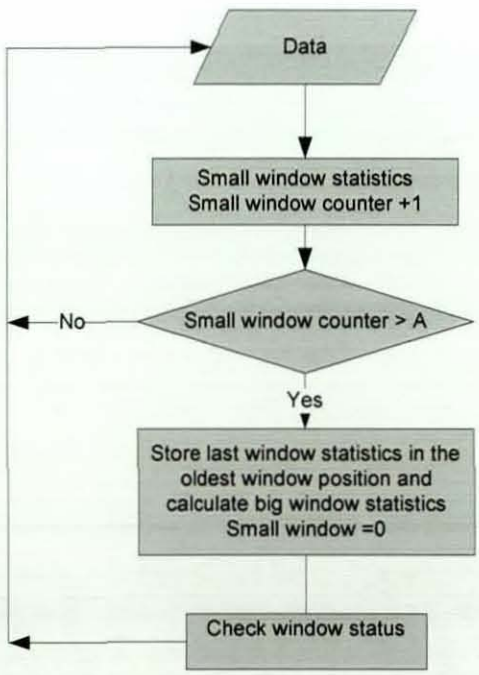

Figure 4-17. Sliding Window Functionality Flowcharts

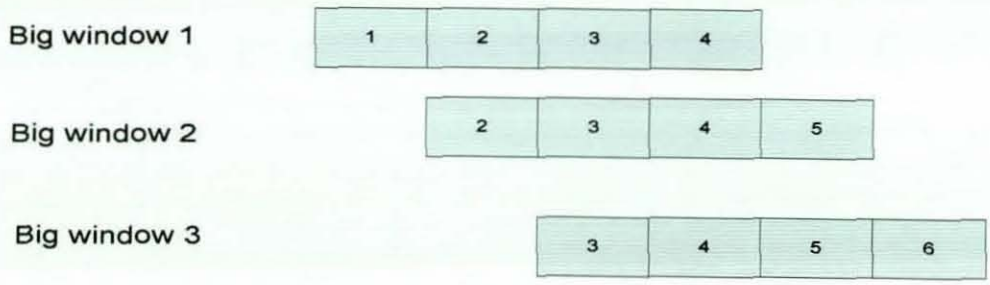

Figure 4-18. Sliding Window Movements in Time

4.4.3.2.3 Flip-Flop Window

This type of window depends on an incrementing counter when the threshold is exceeded and reset if it does not satisfy requirements, as shown in Figure 4-19. The main advantage of this window is its low memory usage and its low response time. However, this window comes with the disadvantage of missing the deviations that temporarily affect the network's functionality.

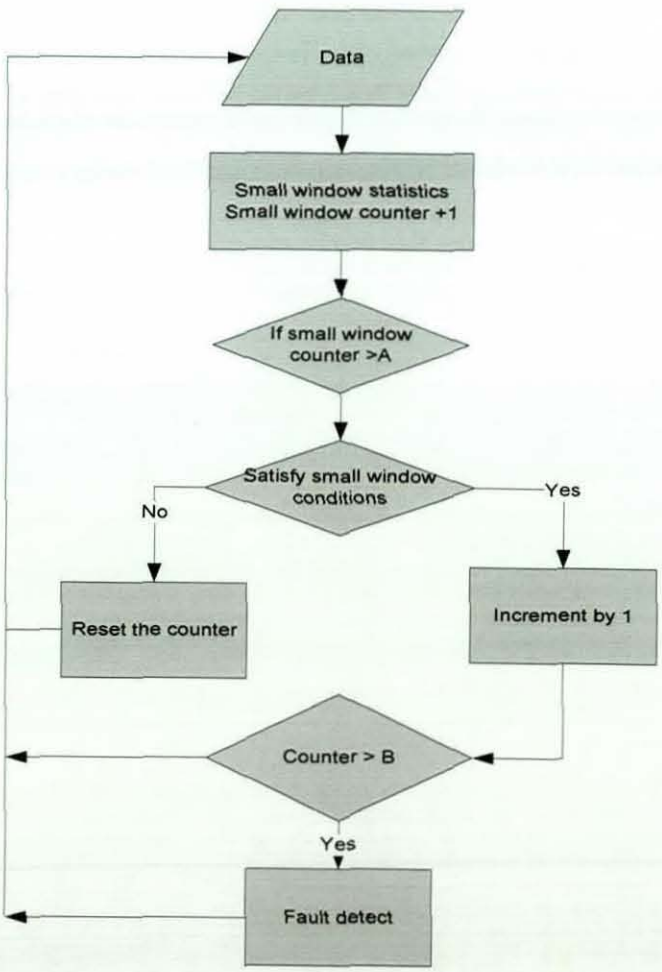

Figure 4-19. Functionality Flowchart of Flip-Flop Window

4.4.3.2.4 Comparisons of Window Characteristics

Table 4-4 shows a comparison between the three monitoring window methods described above, programmed with the algorithm on a Mica2 node and using PowerTossim as a tool simulator. The experiments were conducted to test the power consumption and memory used in a neighourhood with the number of nodes ranging from 2 to 16 within the same range and sending their reports to the sink which was a single hop from them. As can be seen from the table, the maximum *RAM* capacity is for the sliding window approach, and the lowest for flip-flop window. *CPU* power consumption is almost the same, however, and the lowest source code complexity is for the flip-flop approach.

	RAM(bytes)	ROM(bytes)	CPU Power[3] in mJ	Number of lines in source code
Periodic window	3654	24518	732.36±6.03	152
Sliding window	3823	24436	731.65±6.25	164
Flip-flop window	3584	24324	733.32± 6.766	139

Table 4-4. Comparison between the Three Window Types

The functionality of the proposed algorithm utilises a combination of the three windows (described above) depending on the type of detection event required. The periodic window was chosen to increase the probability of detection in events concerning connectivity instability, while a sliding window was used to detect neighbourhood malfunctions and neighbourhood degradation in the accuracy of collected data. This is to reduce the effect of these deviations on the network's functionality, and to increase the confidence of the algorithm's detection and the accuracy of the collected data. Finally, a flip-flop window was used to detect dead nodes, node malfunctions and coverage degradations as a result of continuous faults.

4.4.4 Message Exchange Module

Figures 4-20 and 4-21 illustrate respectively the flowchart and the pseudo-code of the module 4 process. The module starts its function by receiving a send request from module 3. Then it checks its neighbours' warning exchange

[3] Confidence interval for power consumption mean at 95% confidence level

memory to ensure that none of the neighbour nodes have reported the same warning at that window monitoring period. If none of the neighbours has ever reported, it sends a message or cancels the request.

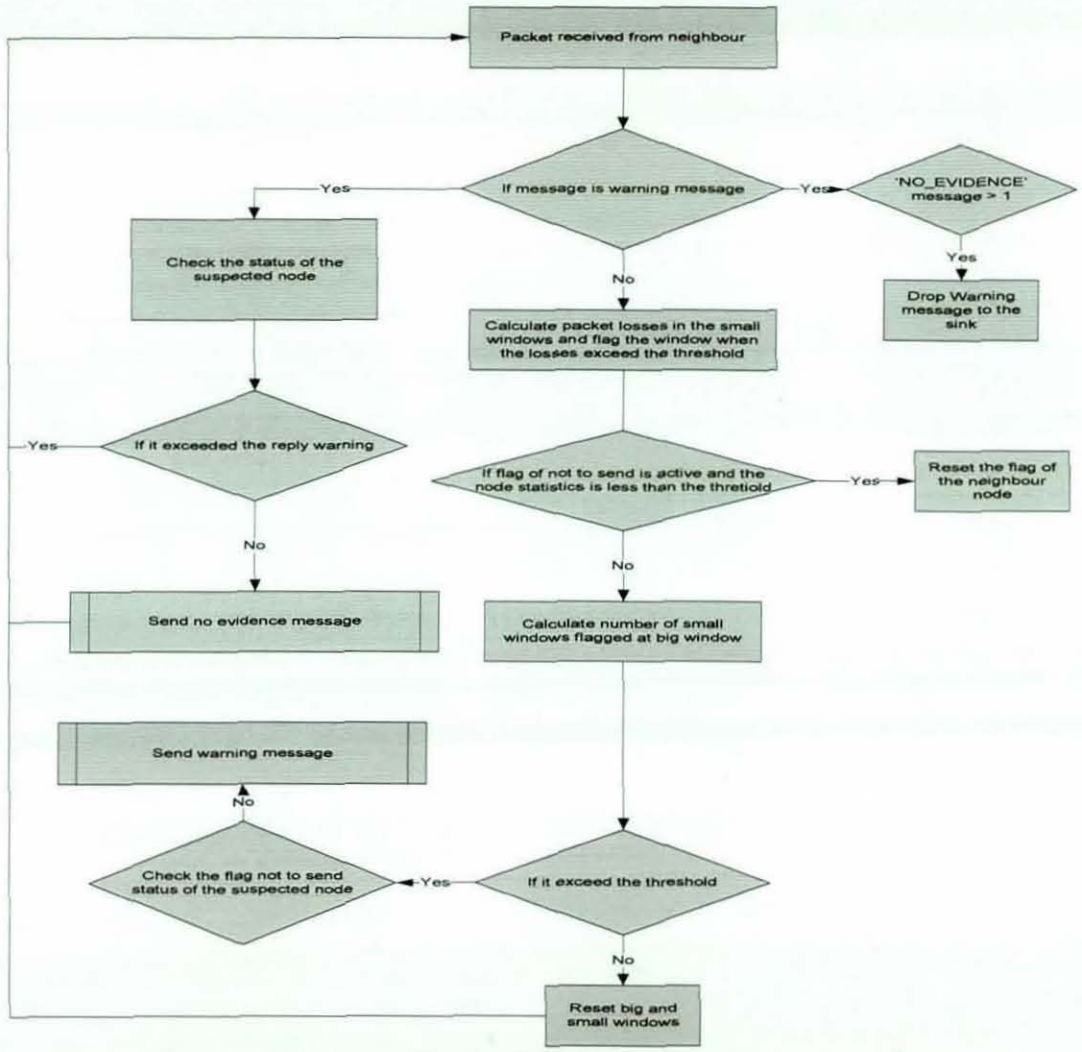

Figure 4-20. Flow Chart of Message Exchange module

Also, this module compares the warning messages received from neighbours with module three statistics. If the flag counter indication from a suspect node is smaller than a threshold (i.e. a level of 30% was set during the experiments), a message will be released indicating that there is no evidence for the existence of the received warning message. However, if the threshold is higher or equal, then the module will cancel any similar warning messages requested from module 3 during that monitoring period. This introduces a passive algorithm detection test in the neighbourhood that ensures the reliability of the warning message; it also rectifies any wrong detections that

occur as a result of losses or other network conditions. Moreover, module 4 reduces the release of warning packets from neighbours by dropping a required send warning message to the sink if it receives more than one *'NO_EVIDENCE_OF_FAULT'* messages. This will save consumption of the multi-hop routing message, reducing it to consumption of the broadcasts in the local neighbourhood. Finally, this module stops reporting suspected neighbours after a certain number of detections and then sends a message to the network user. It also self-configures the network after isolating the suspected node from the network's functionality so that the network will not depend on it.

1: Receiving neighbour warning
- a) Check received warning with the same module 3 counter of the reported node.
- b) IF module 3 counter < 30%
- c) Release *'NO_EVEDENCE_OF_FAULT'* message
- d) ELSE flag the stop sending of the same message from the node at this monitoring time.

2: Receiving module 3 request
- a) Test stop flag of received request warning
- b) IF flag = 1 discard message
- c) IF send message repeated 3 times send *'FAULT_MESSAGE_STOP'* message and flag stop fault counter.
- d) ELSE send the requested message by module 3.

3: Testing warning packet release
- a) IF detected fault returns to normal reset the same fault counters, send *'FAULT_CLEAR'* message and recalculate protocol tables.
- b) IF step 2 and 3-a alternate for the same fault three times in a predefined monitoring window, the module sends an *'TOPOLOGY_UNSTABLE'* message to report the detection and flags a permanent fault counter to stop reporting the same fault.
- c) If the number of *'NO_EVIDENCE_OF_FAULT'* messages in the neighbourhood exceeds 1, then the warning message intended to be sent to the sink is dropped.

4: By the end of the predefined period reset all counters.

Figure 4- 21. Warning packet exchange module Pseudo-code

After the suspected node is isolated from the neighbourhood functionality, the algorithm continuously monitors the suspect node for a time and, if it returns to the normal neighbourhood functionality, module 4 releases a *'Fault_ Clear'* message and sends a request to the protocols to recalculate their tables. If this suspected node enters this stage three times within a predefined period,

the algorithm considers it as a frequent problem due to highly dynamic topology or coverage. It then sends to the sink an *'UNSTABLE_FAULT'* message and stops reporting the same fault after isolating it.

4.5 Detection Confidence of the Algorithm

The detection confidence of the proposed algorithm is controlled by the monitoring window size and validity tests on the collected data. This is due to variations in values at every event as a result of changes in the number of measurements per event, changes in the number of losses per event, and changes in the degree of closeness between measurements. The following sections discuss confidence control and its tradeoff.

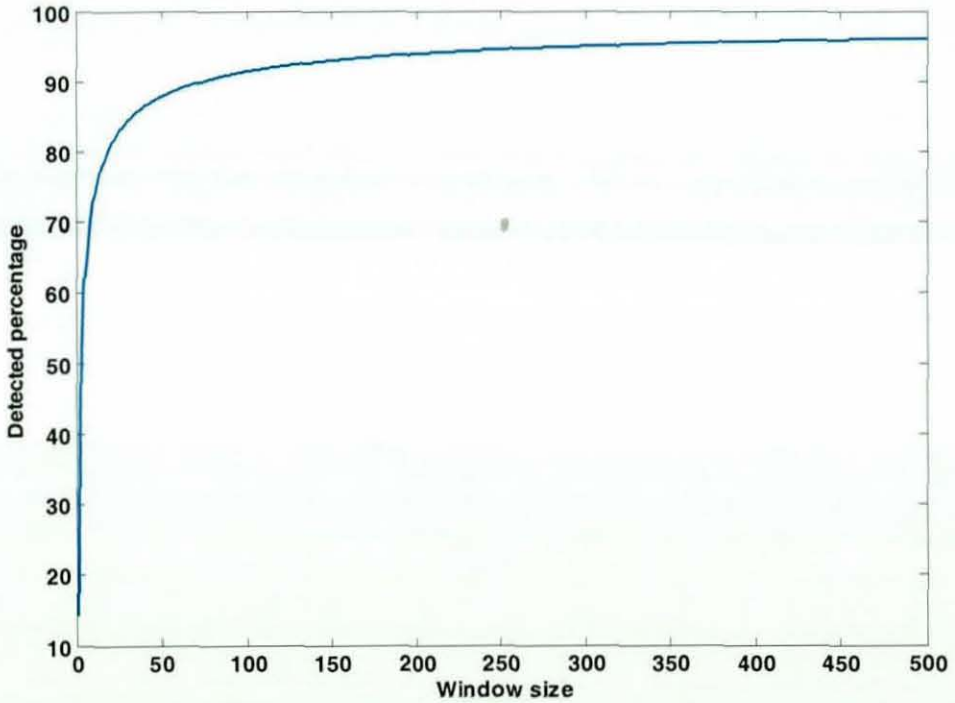

Figure 4-22. Confidence Detection with 70% Threshold

4.5.1 Deviation Measurements and Monitoring Window Selection

Measuring how true the detection of a deviated sensor output is can be carried out with the help of monitoring windows and their thresholds. The setting of their size depends on the level of detection confidence required. For example, Figure 4-22 illustrates the detection of different sizes of monitoring

window with outliers at a level of 70% in a data set. This simulation experiment used the statistical analysis of one sample situation, discussed in [51], and an assigned monitoring window threshold of 70%. The figure shows that as the window size increases, the detection of outliers increases up to a maximum level; after that, it remains constant for any larger size of monitoring window (e.g. for 100 samples, as shown in the figure).

In addition, the size of the monitoring window depends on the percentage of outliers in the data set. For example, Figure 4-23 illustrates the detection of outliers for different sizes of monitoring window. The simulations in these experiments also used, for one sample situation, the statistical analysis discussed in [51] and a monitoring window threshold of 95%. As can be seen from the figure, the confidence in detecting outliers increases as the window size increases up to a point, after which the detection remains essentially constant.

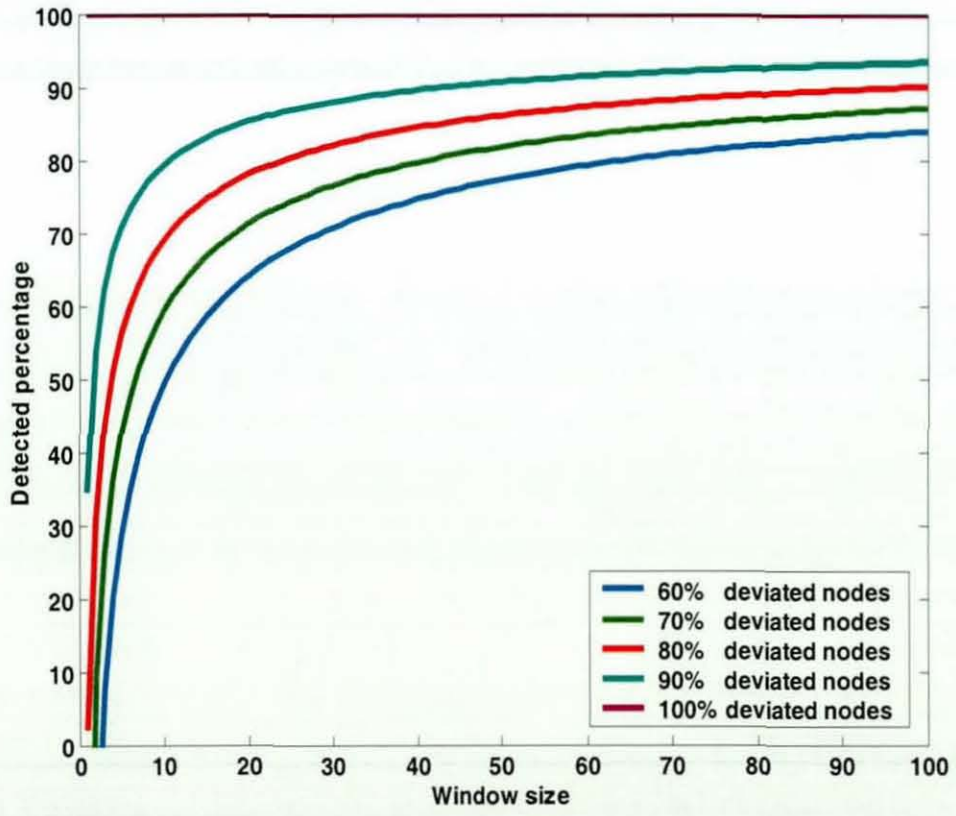

Figure 4-23. Faults Detection Percentage for Different Monitoring Window Sizes and Outlier Percentages

In order to calculate the probability of monitoring window detection for the outliers in a data set, a discrete probability distribution, describing the number of successes in a sequence, was drawn from a finite population without using replacement (i.e. it is a hypergeometric method). Thus,

$$f(x) = \frac{\binom{M}{x}\binom{N-M}{n-x}}{\binom{N}{n}} \qquad (4.3)$$

where N is the sample size for the total data set, n is the sample size of the monitoring window, M is the total number of deviated data samples in the data set, N-M is the total number of healthy data samples in the data set, and x is the minimum number of deviated data samples detected at that monitoring window (i.e. the monitoring window threshold). Then,

$\binom{M}{x}$ is the different ways used to obtain x deviated samples;

$\binom{N-M}{n-x}$ represents the possible ways to fill the rest of the monitoring window samples with healthy data;

$\binom{N}{n}$ calculates the possible ways of selecting n sample windows from the N samples size data set, while

P (at least x deviated data samples detected at a window)

$$= \sum_{k=x}^{n} number\ of\ \text{detected}\ deviated\ data\ samples$$

$$= \sum_{k=x}^{n} \frac{\binom{total\ number\ of\ deviated\ data\ samples}{x\ deviated\ data\ samples}\binom{total\ number\ of\ none\ deviated\ data\ samples}{monitoring\ window\ sample\ size - x}}{\binom{\text{total}\ \ size\ \text{of data}}{monitoring\ window\ sample\ size}} \qquad (4.4)$$

This is the formula for detecting the probability of different distributions of deviations in a monitored window. Figure 4-24 shows the results based on formula (4.4) (for different monitoring window sizes with a 50% detection threshold). The figure illustrates that between 40 to 60% deviated data with a size 2 window has the highest probability of detection. For greater than 60% deviations the size 8 window has the highest probability of detection. Figure 4-25 shows that the probability of not detecting a deviation in the algorithm with the size 2 window is around 0.51.It is reduced to 0.19 for a size 8 window.

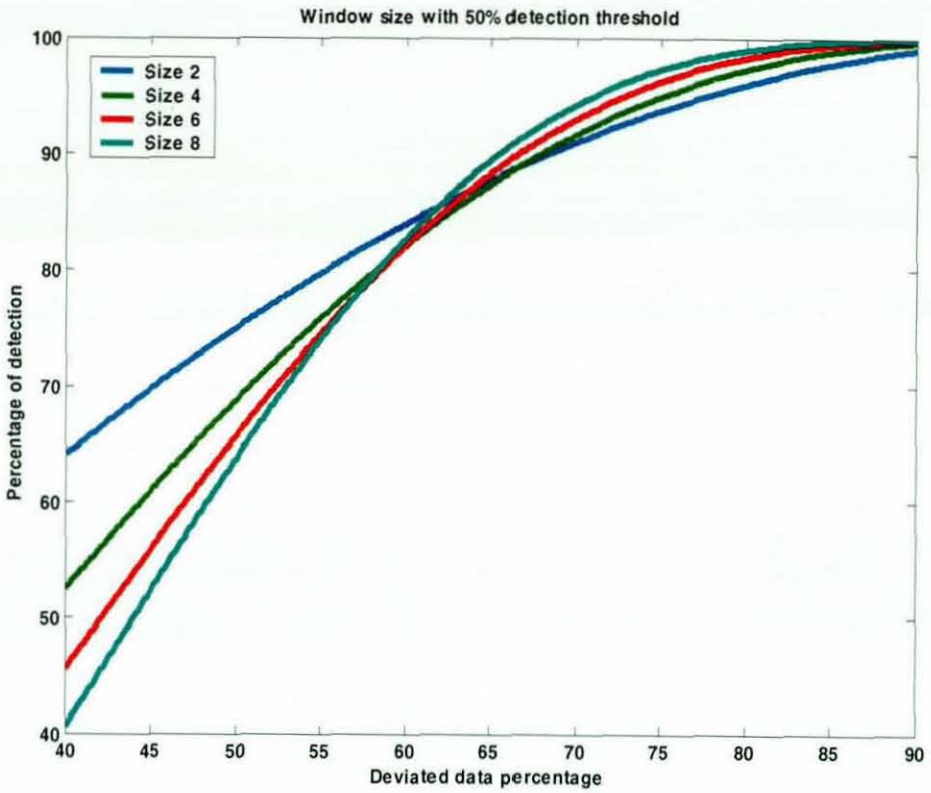

Figure 4-24. Percentage of Outlier Detection with 50% Threshold for a Small Window Size

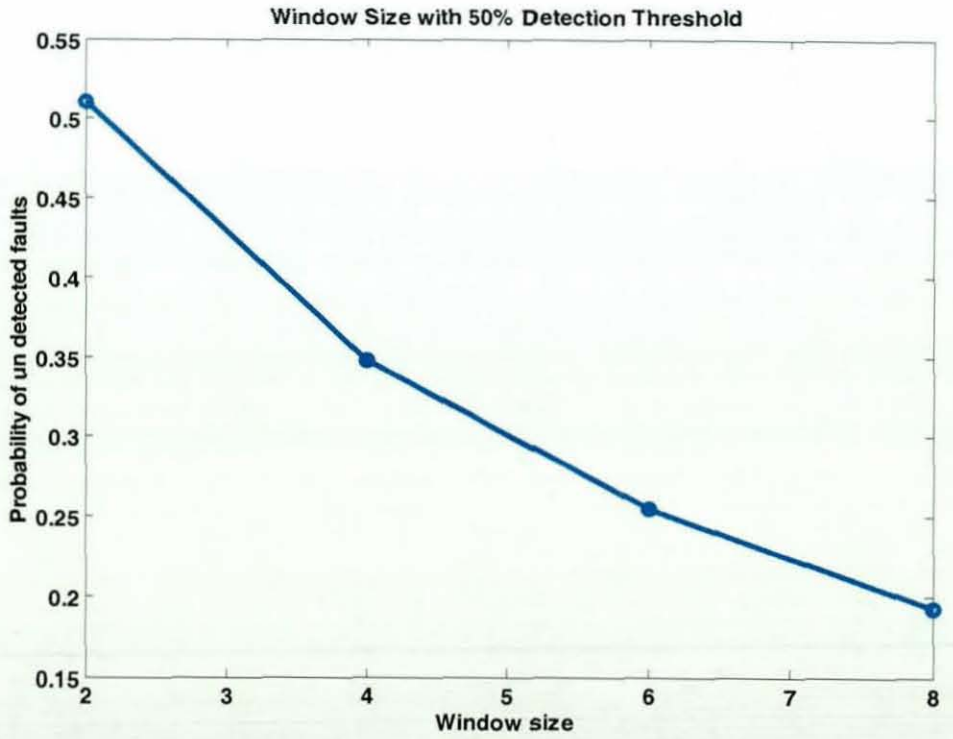

Figure 4-25. Probability of Errors in Fault Detection with Various Window Sizes

In figure 4-26, when the assigned detection threshold is set to 70%, the highest probability of detection is for a size 2 window while the lowest is for a size 8 window (for all the deviated data ranges; i.e. 40-90%).

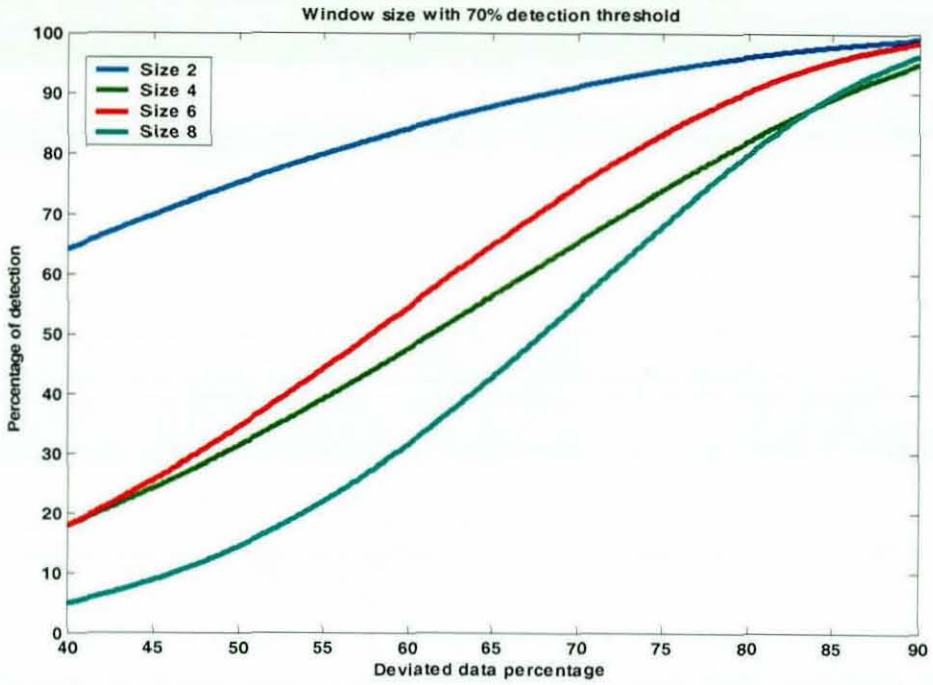

Figure 4-26. Percentage of Outlier Detection with 70% Threshold for a Small Window Size

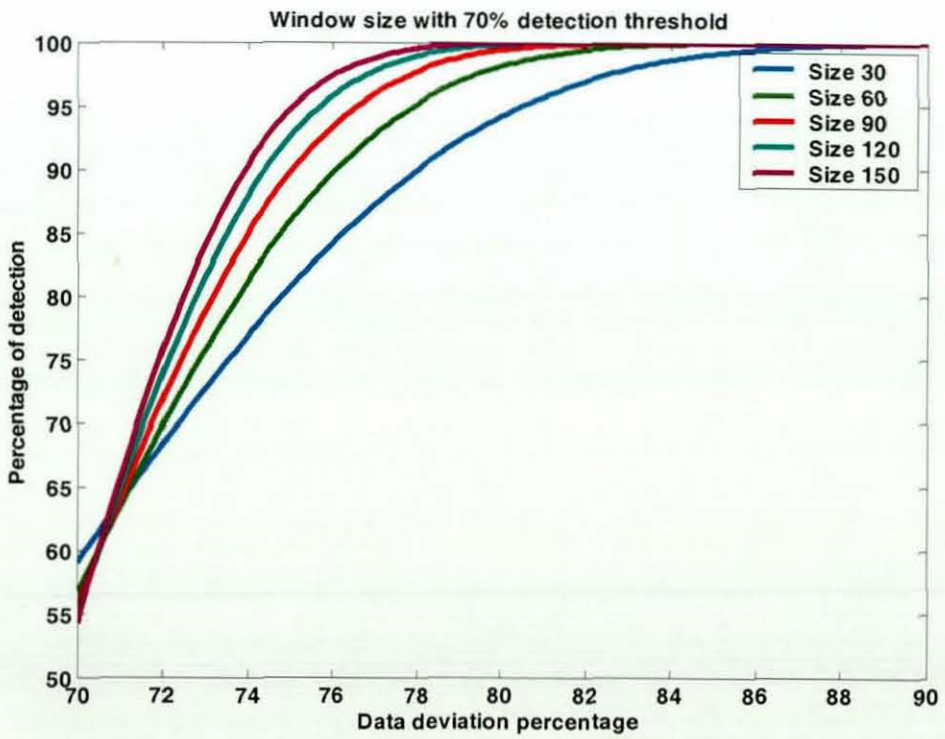

Figure 4-27. Percentage of Detection in Large Window Size

On the other hand, with large window sizes, the highest probability of detection is for the window with the largest size. This has been shown in Figure 4-27. From Figure 4-28, we can see that the window with the largest size shows the least error in deviation detection.

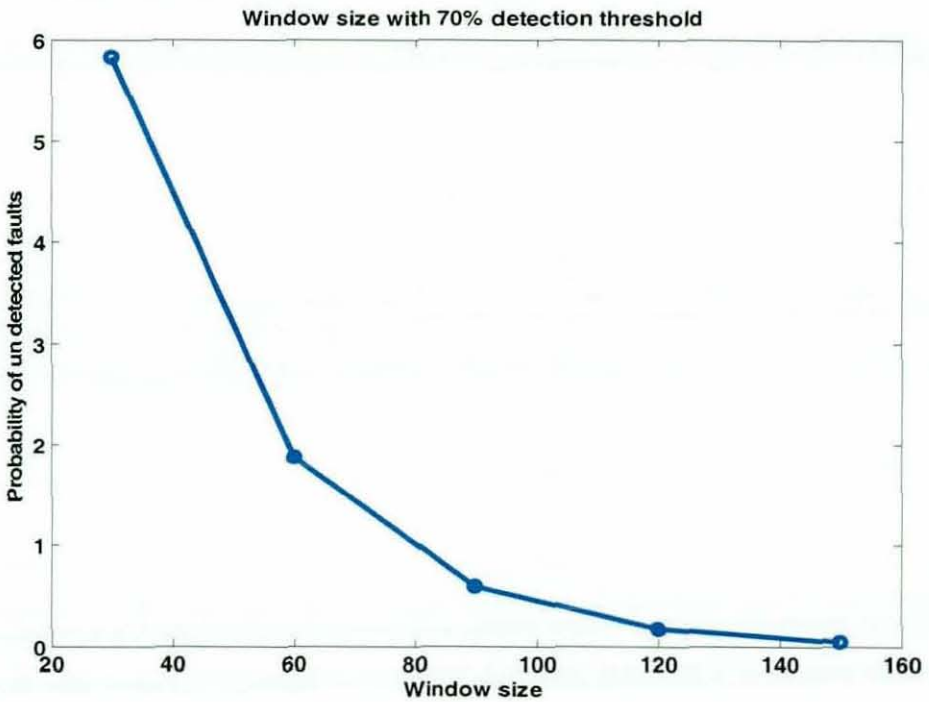

Figure 4-28. Probability of Errors in the Fault Detection for Different Window Sizes

Finally, Figure 4-29 illustrates the probability percentage of the algorithm's error detection versus different window sizes for different numbers of readings. The figure shows that detection error variations depend on the number of readings where, with a window size of 11, if the number of readings is 1, then the errors total 19%. However, if the number of readings is 7, then the probability of detecting an error is almost 0%; in such a case, the algorithm's response time increases.

From the above discussion, we conclude that there is a tradeoff between the selection of the monitored window size and its threshold. Also, these windows can be dynamically set, depending on the number of readings, to increase the detection. Another way for optimizing the monitoring window period depends on the network application goals as discussed by Araujo *et al.* at [113].

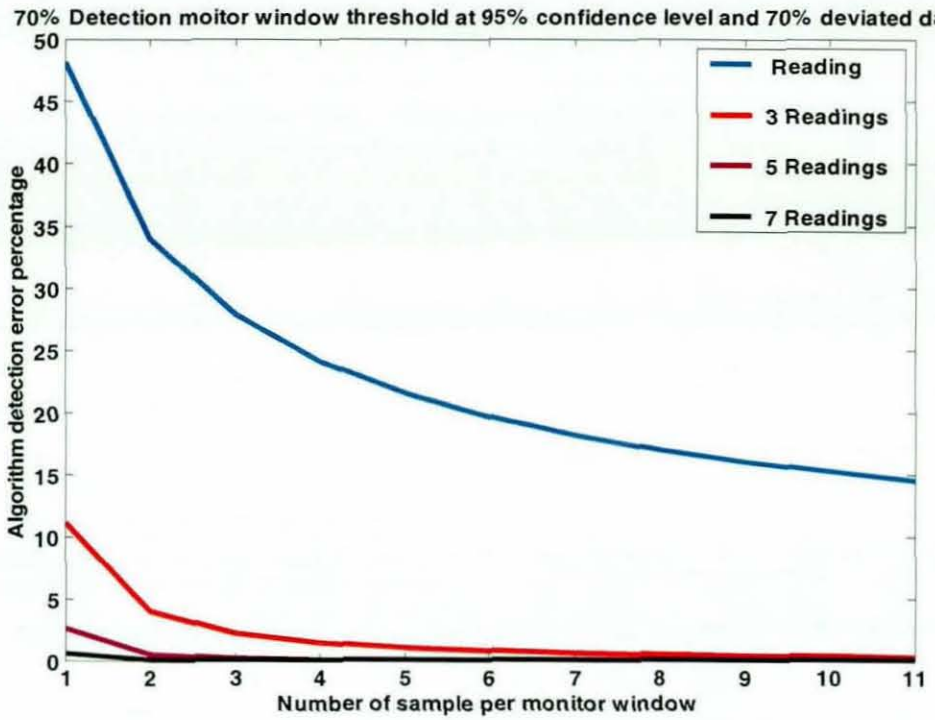

Figure 4-29. Percentage of Errors in Fault Detection with Various Window Sizes and Neighbour Node Numbers

4.5.2 Dead Node and Monitoring Window Selection

Unreceived packets from a neighbour can be due to several reasons: e.g. signal strength fading (leading to low signal to noise ratios over long distances); the environmental interface; packet collisions between multiple transmits; asymmetric links[4]; power transmission range; receiver hardware; sleeping nodes; and leaving the hearing range. These factors affect many algorithms and cause the protocols to function inefficiently.

The proposed algorithm tracks neighbour nodes and their losses; if it fails to hear from neighbours for a certain period, it releases a suspected dead node message. However, due to the highly dynamic topology and high losses in *WSNs*, the monitoring window should be bigger than other monitoring windows, otherwise it will cause a lot of false detections.

The dead node monitoring window depends on the responsiveness of the algorithm's detection. Figure 4-30 illustrates different monitoring window sizes

[4] Sending and receiving loss rate differ.

and expected errors in the detection of dead nodes for different packet losses. When the time for receiving a packet from a neighbour expires, the probability of that packet not being received due to loss decreases exponentially with time, as discussed in [11]. For example, if the losses in the network total 90%, then the percentage of packets not being received after 20 samples because the packet has been lost in the network is 12%, as shown in Figure 4-30.

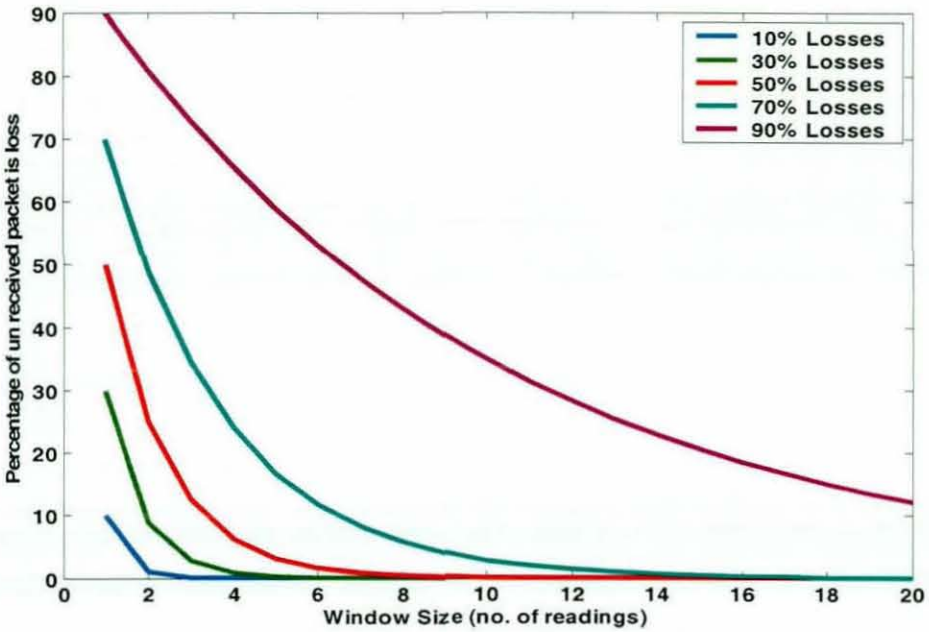

Figure 4-30. Relation between Expected Percentages of Packet Losses and Window Size for Different Outliers

If this is grouped into one big window, with readings from 10 small windows, the false detections are reduced to 1% with 90% losses for six big windows in a network, as shown in Figure 4-31. This reduction in false detections results in a higher detection latency which is determined by the size of the big window.

The percentage of losses in the network can be used to assign the dynamic "aliveness" of window size in order to increase the detection confidence; this comes with the usage of more node memory, however. Hsin *et al.*, in [61], considered different approaches for detecting dead nodes, using a two-phase timeout system with distributed monitoring. This monitors dead neighbour nodes but Hsin used a greater level of message exchange to confirm the death of the neighbour. The main drawback of his algorithm was the

exchange of the request messages between neighbours at the second phase window. In the algorithm proposed in this study, this has been solved, as explained in the section concerning the message exchange module, so that, if the neighbours do not agree with a warning, the algorithm replies.

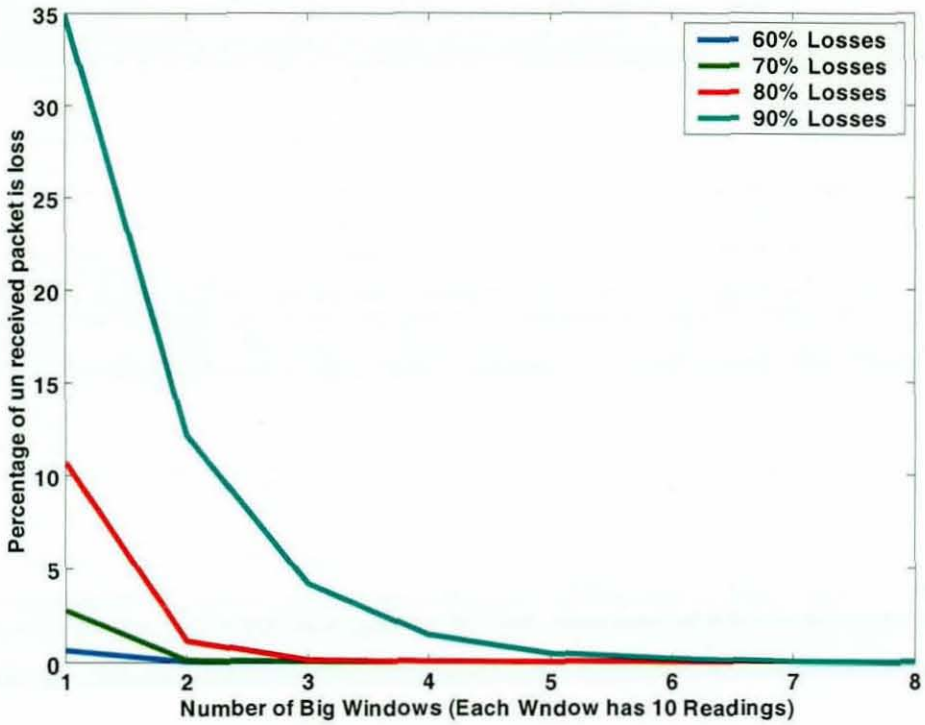

Figure 4-31. Big Window Probability of Neighbour Packet Loss with Readings from 10 Small Windows

4.5.3 Adjusting the Algorithm's Confidence Detection Depending on Collected Data Validity Tests

The probability of false alarms released by the proposed algorithm increases as a result of losses, the impact of external factors on nodes, and different responses by nodes to changes in a phenomenon's characteristics. These false alarms can be reduced by increasing the size of the monitoring window but this also increases the impact of the fault and the algorithm's detection response time especially if the application's reporting rate is low; as discussed in [62].

In order to increase the algorithm's detection confidence in such applications, the algorithm has been designed to use data validity tests as confidence

parameters. These tests depend on a range of physical changes in the monitored phenomenon, the redundancy of readings between neighbours, zone of sensor operation, error of median calculation and the degree of closeness or conflict between neighbours' readings. Testing a combination of these parameters through empirical experiments showed a reduction of false alarms with some negative faults. This is discussed in Chapter 8.

Also, confidence tests can be organised into an influence diagram; i.e. cause and effect; that represents the detection of faults in a *WSN*. Thus, this can replace a monitoring window which has a quicker time response and a lower impact on network functionality. (This work has been left for a future extension of the scope of this study).

4.6 VMBA with a Decision Classifier

In addition to the *VMBA* algorithm's analysis (discussed earlier), a classifier can be added to categorise the cause of detected deviations. This can be done by comparing each node to a neighbourhood median and other neighbour readings to check the location of readings that depend on other readings and medians.

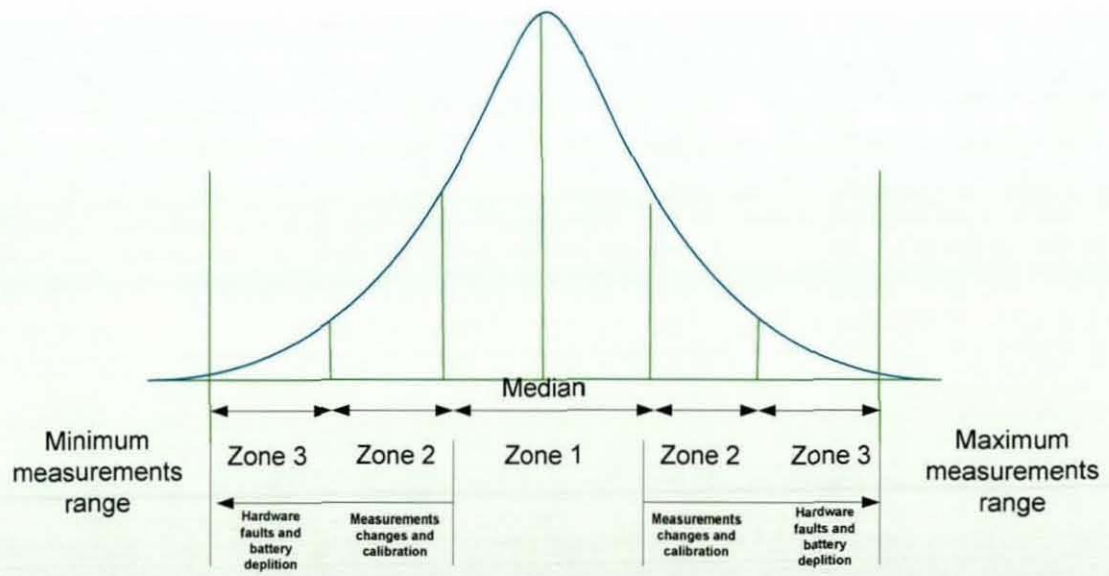

Figure 4-32. Probability of Node Readings in Different Zones in Classifier Detection

Figure 4-32 shows the probability distribution of deviations from the neighbourhood's calculated median of received readings. The measurement value in Zone 1 is accepted as a correct reading. This zone depends on the network's deployment and coverage, as well as sensor characteristics such as resolution, repeatability, equipment accuracy, long-term stability and response time. It also depends on the accuracy the user defines between the nodes, the noise level of the phenomenon, and the measured changes in the phenomenon's characteristics. (In simulation experiments, this zone is assigned 5% of the median value.)

Zone 2 is where measurement changes can occur due to external effects on the nodes in the neighbourhood and calibration settings. Its value depends on the dynamic response of the sensor node in the transient stage. Real world data simulations have shown that detection in this rage should be received from more than one node. If one reading only appears, this indicates that this node is either faulty or the network does not having sufficient coverage to detect a change in the phenomenon or the calibration is set inaccurately.

Zone 3 is the fault area which deals with permanent faults. At the beginning of this zone, faults may occur due to environmental change or, at the end, due to node faults. Readings in Zones 2 and 3 are permanent or temporary. In Zone 2, temporary readings will be due to environmental, software bugs and/or measurement changes, while permanent deviations will be due to changes in calibration and/or measurements.

The fourth zone deals with measurement values greater than the maximum and smaller than the minimum ranges of the sensor. For example, for '*SHT1x/SHT7x*' sensor [88], the maximum is $123.8C^{O}$ and the minimum is $-40C^{O}$.

The Guassian distribution shown in Figure 4-32 is only a representation of the predictable manner change of sensed measurements in space. The algorithm functionality tracks the expected phenomenon value at the end of the node

receiving range. This means that we can use most of the distributions (that are changing in predictable manner) with the algorithm without affecting any of its functionality.

Simulation experiments show that the classifier method can distinguish, to a certain percentage, as discussed in Chapter 7, between different types of fault. However, it was not possible to implement this on the testbed due to the size of memory it requires. (The existing platform could not provide this.) As a result of this, the zones were reduced to two (i.e. 1 and 2) and these were compared with the neighbourhood median value. This modification enabled the proposed algorithm to isolate changes in the environment and the phenomenon, and to isolate coverage problems.

4.7 Summary

This chapter discusses the functionality of different modules of the proposed algorithm and the theories behind them. It lists metrics used and the detected events. Moreover, it explains those methods that control the proposed algorithm's detection confidence and then describes the extension of the classification function in the algorithm's analyses.

Chapter 5 Modifications to the VMBA Algorithm

5.1 Introduction

As discussed in Chapter 2, a *WSN* protocol stack can be divided into two main parts: a high network level part that represents the application's protocols, and a low network level part that represents communication protocols. For example, in applications that construct clusters depending on the nodes' correlation, the communication tree will be constructed depending on the interconnection between these clusters and the sink. The low-level network functionality depends on and is controlled by the high level protocols. As a result, there are two methods of implementing the algorithm in *WSN* protocol stacks. The first method implements components of the algorithm's modules in both the application and communication layers, depending on where the algorithm metric parameters are available. The second implements these components in the communication layer and pushes down the application's measurements from the application to the communication layer. Then sends them to neighbours in routing update packets, or beacons, or path/link advertisements.

This Chapter discusses both methods together with their advantages, drawbacks and the modifications that are then required in the algorithm's structure.

5.2 The Algorithm's Implementation in both the Application and Communication Layers

The proposed algorithm can be implemented in both the application and communication layers such that its module components operate at the locations where they get the required parameters for analysis. The algorithm's different modules are linked together by program components that are responsible for the algorithm's overall functionality, as will be discussed in Chapter 8.

The main advantage of this implementation method is that it offers high confidence levels in the algorithm's detection due to the high availability of

data that are collected directly from the application layer. However, this then necessitates that the proposed algorithm's structure is changed for each application. Moreover, with event-driven and query applications the proposed algorithm will not detect degradations in the network's functionality unless it first satisfies the application's reporting threshold value.

The following three sections discuss the modifications that are required for different *WSN* applications in this implementation.

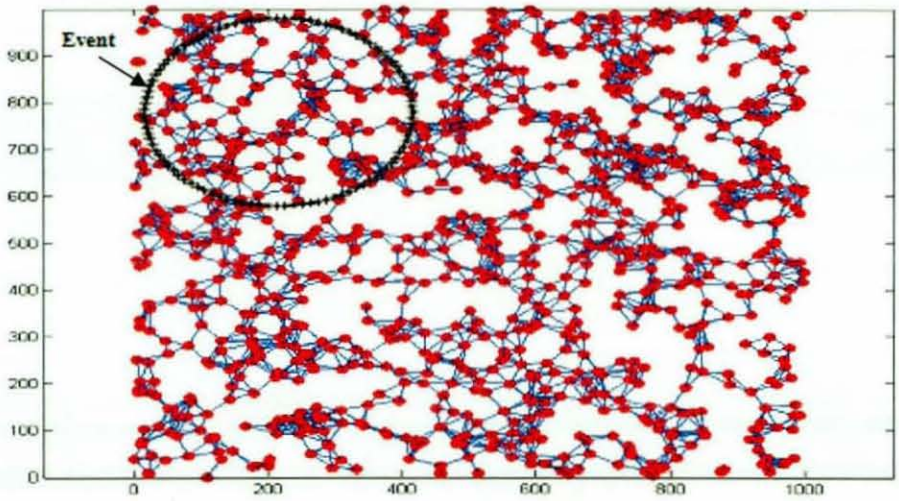

Figure 5-1. 1000 Sensor Nodes Randomly Distributed in a 1000 X1000 Meter area and their Connectivity

5.2.1 Continuous Reporting Application

With this application, all nodes report their measurements continuously. The proposed algorithm works without requiring any modification to its structure. Changes need to be made to the measurement accuracy metrics as these affect certain of the application's measurements such as value and delay. The algorithm in this type of application is affected by the reporting rate, the number of neighbours, and the degree of the correlation between their measurements, as will be discussed in Chapter 6.

5.2.2 Event-driven Applications

With this type of *WSN* application, only nodes that reach a certain condition are triggered by the phenomenon to report the event. Figure 5-1 shows 1000

random distributed nodes over a 1000X1000 square metre area using *MATLAB* tool. Each of these nodes has a 50 metre sensing and transmitting range. If an event occurs in the circled area, as shown in the figure, and 10% of the randomly distributed nodes are deviated (i.e. 68 outside and 32 inside the event), the proposed algorithm was found to detect a total of 124 deviated nodes including 36 positive false and 12 non detected faults. Most of the algorithm's false detections occurred at the event boundary where there are some nodes with measurements that did not exceed the assigned application's event threshold. Because of this, the algorithm, in its existing structure, will release a large percentage of false detections and will not detect certain deviated nodes.

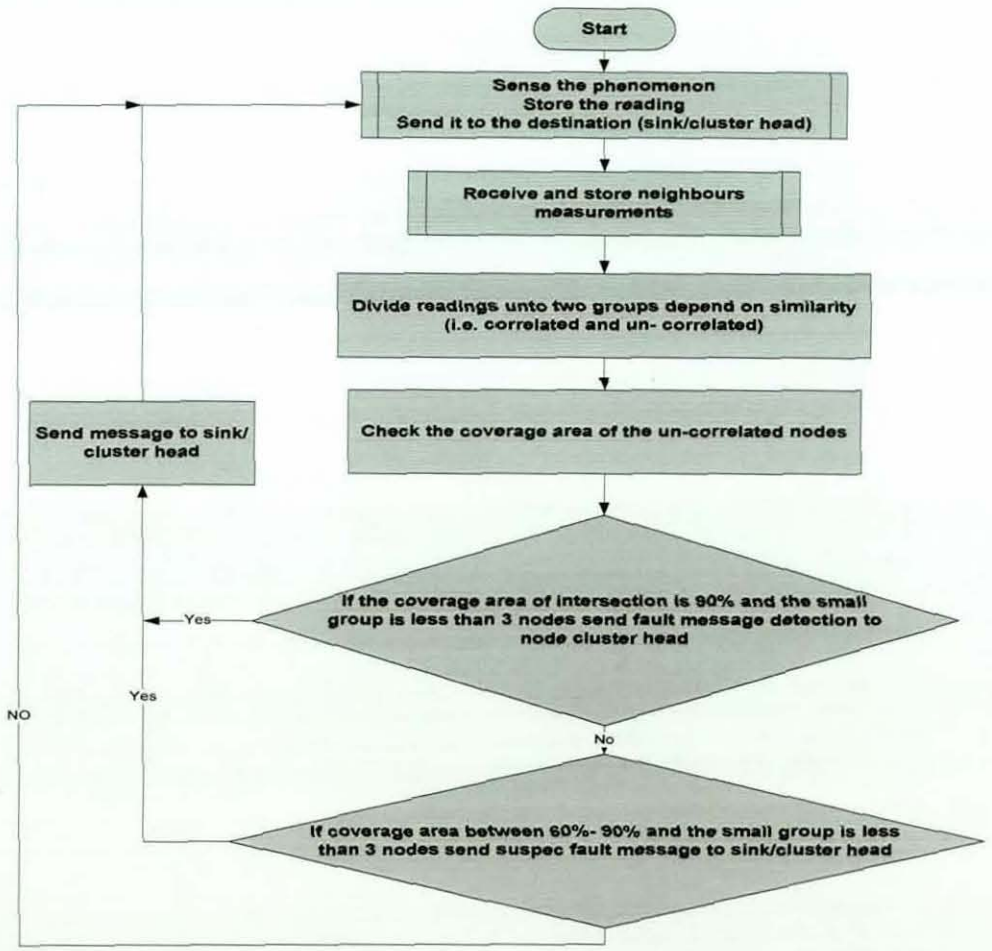

Figure 5-2. Functionality Flowchart in an Event-Driven Application

To reduce such event boundary errors, the proposed algorithm's structure was modified, as shown in Figure 5-2, utilising knowledge of the nodes' locations in this type of application.

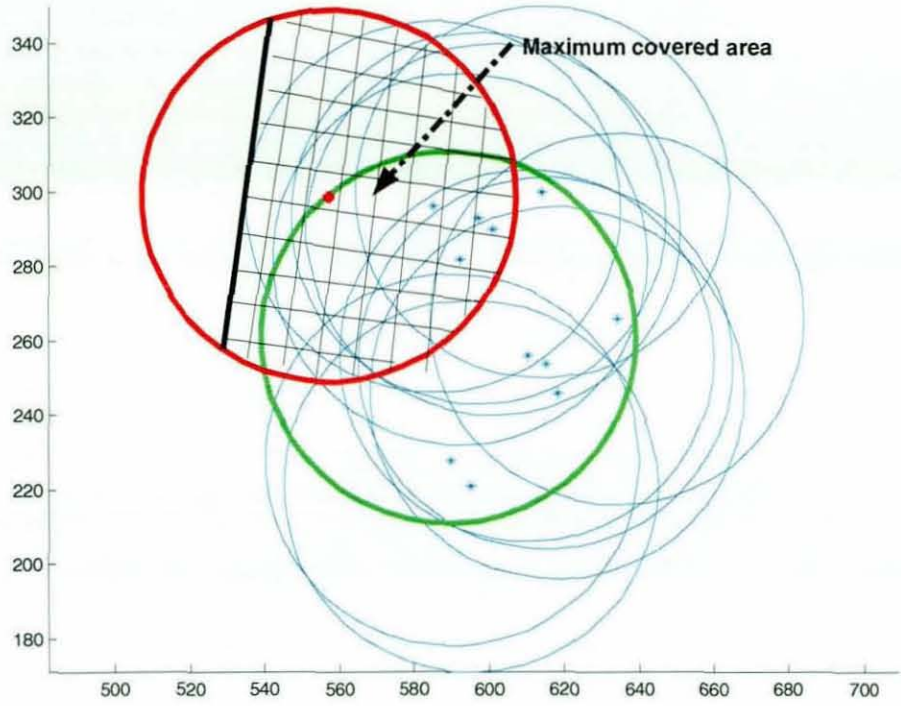

Figure 5-3. Zone Detection Method

In this structural modification, each monitor node considers itself to be at the center of the virtual evaluation quadrant, as shown in Figure 5-3, while the location of its neighbours depends on their distance from the centre. Then, the algorithm divides the readings from these nodes into two groups, depending on their similarity. This is followed by the process of selecting the largest group as correlated readings and the smallest as non-correlated. If the number in the non-correlated group is more than two, then this is considered to be an event. Else, the algorithm checks the physical coverage of the correlated nodes to the nodes categorised as suspect, as shown in Figure 5-3[5]. The coverage probability of the correlated group to these suspected nodes covering area is calculated using the formula $\rho = \frac{B}{A} = 2R^2 \arccos(\frac{d}{2R}) - \frac{d\sqrt{4R^2 - d^2}}{2}$, where R is the sensing radius, d is the distance between the two nodes, the derivation of which was explained in [56][6].

[5] The red circle indicates the coverage of suspected nodes, green is the monitoring node coverage, and blue the coverage of other neighbour nodes.
[6] Please note that d is fixed between the two nodes because the nodes are static.

This is done by using the above formula to check the greatest level of probability in the monitored area, where the intersection occurs between nodes categorised as correlated and suspect nodes. If the area of intersection of maximum probability is 0.9, then the detection is confirmed as a deviation and the algorithm releases a 'DEVIATION_DETECTION' warning packet of suspect nodes. If the probability of suspected node coverage is between 0.6 and 0.9, then the algorithm releases a *'SUSPECT_DEVIATION'* warning packet. Of these, 27 were in the suspected category, five were false positive detections and only two were non detections.

The algorithm threshold probabilities selected depend on the results detected from repeating the above experiment (i.e. described in Figures 5-1 and 5-2) for different probability thresholds for fault and suspect detection. Figure 5-4 shows the average detection for the modified algorithm. The figure shows that the algorithm's detection rate increased as the threshold increased; this went hand-in-hand, however, with an increase in false-positive detections. Also, the figure shows trade-offs between false-positive and false-negative detection where negative detection tends to 0 at 0.9 coverage probability and very low between coverage probability 0.6 and 0.9. Depending on these values, the modified event driven algorithm sets the threshold for 'DEVIATION_DETECTION' and *'SUSPECT_DEVIATION'*.

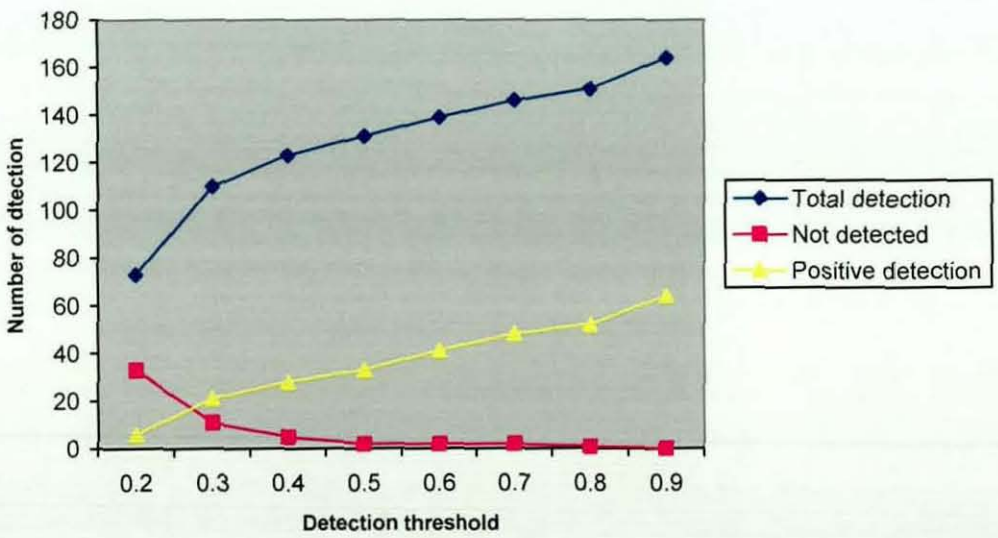

Figure 5-4. Experiments on Number Detections with Different Thresholds

Figure 5-5 indicates the confidence interval of the detection of deviations in the proposed algorithm with the modified structure (i.e. Figure 5-2) for 1000 randomly distributed nodes over an area of a 1000X1000 square metre, each with a 50 metre sensing and transmitting range, using *MATLAB* as a tool. The figure shows that, as the percentage of deviated nodes increases, the algorithm's detection also increases but with a subsequent increase in the number of negative false detections while the positive false detections remained almost constant. In addition, the figure shows that the detection confidence interval of the algorithm varied to a great extent as the number of deviated nodes increased because of the event size, the type and location of faults, and the number of healthy neighbours. Therefore, the detection of the algorithm in this type of *WSN* application depends on the position of the event's occurrence and its size when the number of wrong detections increases at the edge of the detected event. This result is consistent with the detection described in [23], [68], and [76].

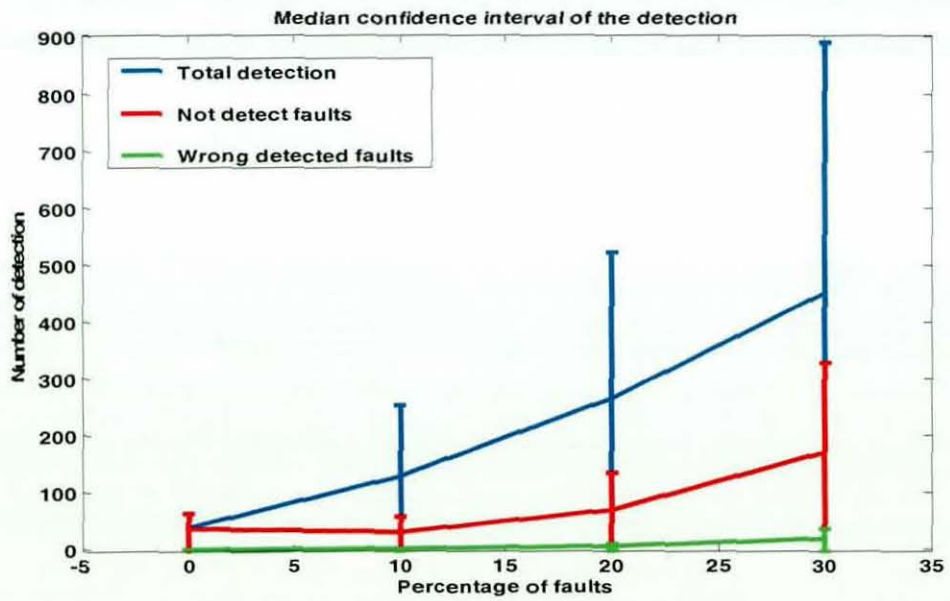

Figure 5-5. Experiment's Confidence Interval for Detection Number with Different Thresholds

To reduce this number of false detections, the algorithm was modified so that it sent a message to consult its neighbours. This failed to add any extra information that could distinguish faulty nodes but consumed more energy because of the exchange of messages between neighbour nodes.

5.2.3 Query-driven Application

This type of application can be divided into single and group non-aggregation/aggregation query applications, as discussed in Chapter 2.

In an aggregation application, the algorithm needs to be modified so that it distinguishes between aggregated and non-aggregated packets in order to compare the measurements with each other. The proposed algorithm was tested by carrying out experiments using a *MATLAB* tool to simulate a network using a combination of distributed election leader and distance-vector routing algorithms (described in [23]), with 500-1400 nodes randomly distributed over a 1000x1000 square metre area, each with a 50 metre communication range (the snapshot of the 1400 node deployment is shown in Figure 5-6) and also using a power consumption model as shown in Appendix A. Results showed that the proposed algorithm detection of deviated nodes in a non-aggregation query application is almost the same as in an aggregation query application, as demonstrated in Figure 5-7.

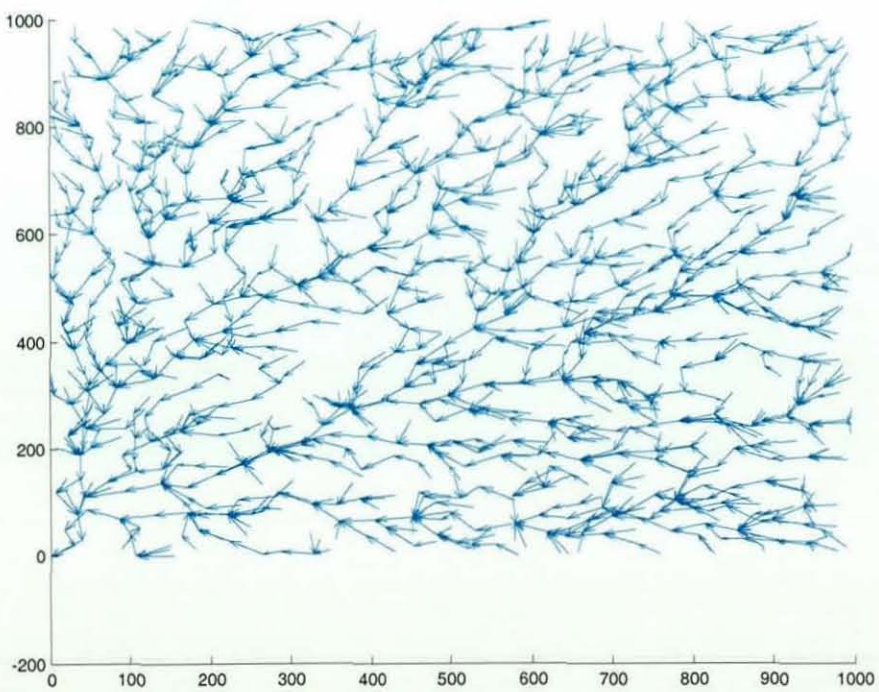

Figure 5-6. Aggregated Nodes in a Random Distribution of 1400 Nodes in a 1000X 1000 Square Metre area Using Distributed Election Leader and Distance-Vector Routing Algorithms

Although these simulations showed that the proposed algorithm's detection with the aggregated query application is slightly less than with the non-aggregated application, it saved around 40% to 50% of percentage of monitoring node reduction when compared to the non-aggregated application, as shown in Figure 5-8. The figure was the result of randomly distributed nodes ranging in number from 500 to 1400 in an area of a 1000X1000 square metre area, each with a 50 metre range and multi-hop reporting to the sink. These experiments used MATLAB code for simulation and the power consumption model shown in Appendix A and using a combination of distributed election leader and distance-vector routing algorithms (described in [23]).The reduction in power consumption was due to the algorithm's use of a cluster head; this reduced the total number of monitoring nodes in the network. However, these aggregated, cluster-head nodes offer less average detection confidence than child nodes because of the smaller amount of raw data received (i.e. this depends on the type and the size of the aggregation).

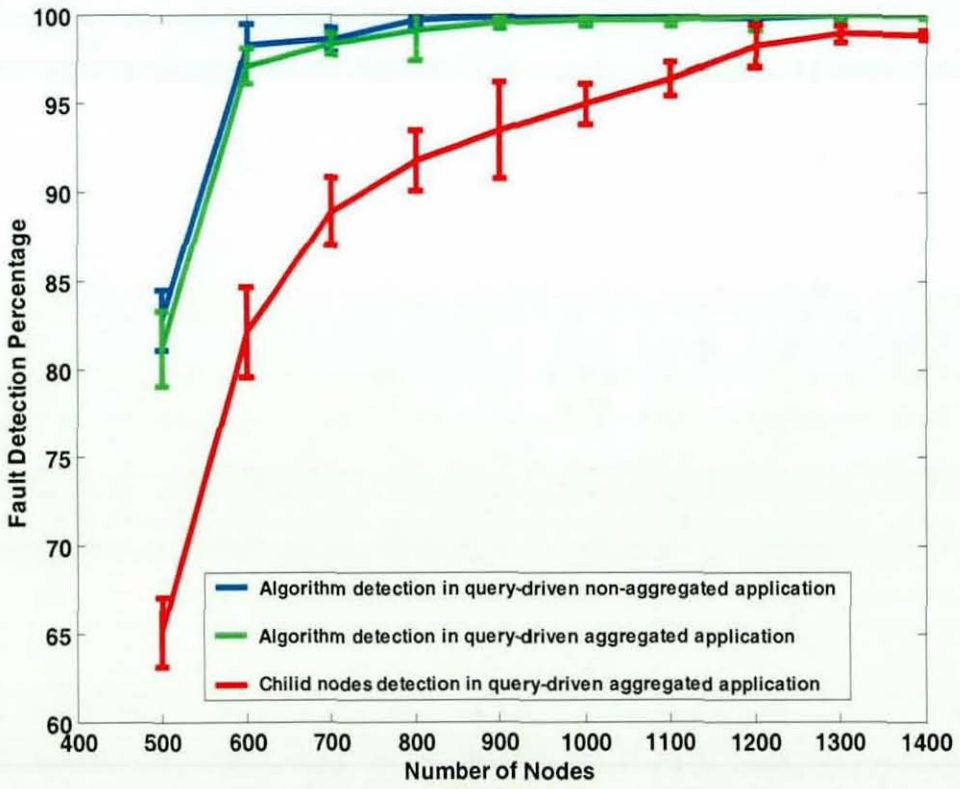

Figure 5-7. Diversion Detection of Aggregated, non-Aggregated and Child Nodes in Aggregated Query Applications

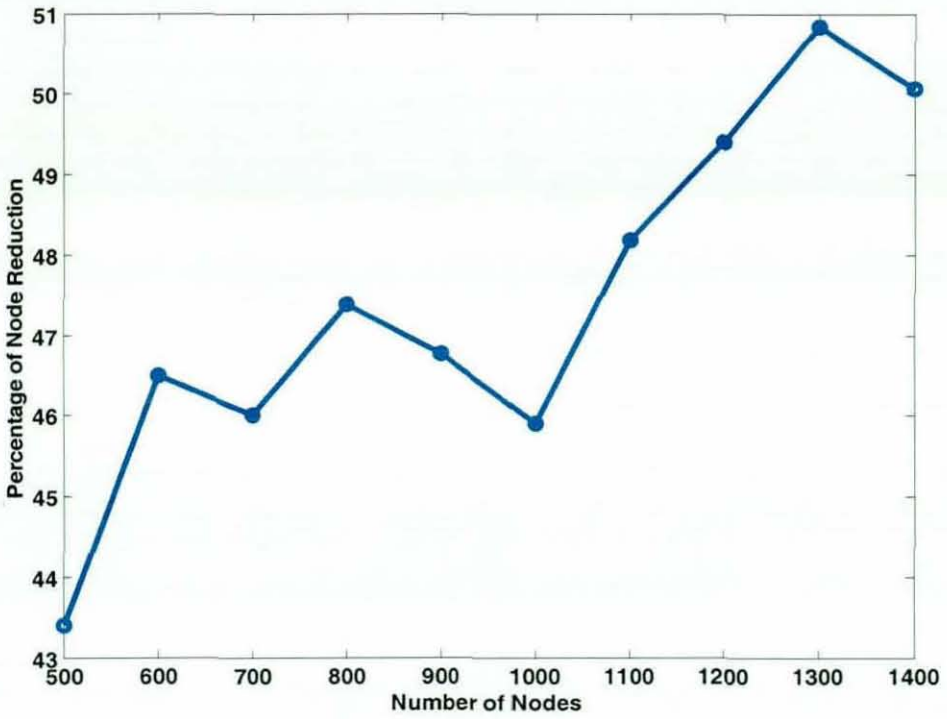

Figure 5-8. The Number of nodes that uses *VBMA* monitoring Algorithm between Aggregated and non-Aggregated Query Applications

On the other hand, in a single query with a query application, there will be insufficient data from neighbour nodes for the proposed algorithm to be able to calculate the correlation and track the changes of neighbours. The proposed algorithm was therefore modified, as shown in Figure 5-9, to enable it to do this by using active monitoring after detecting a change in neighbour nodes. When the application requires a single query from a node, it sends its reading to the sink. If the parent node then detects a deviation from its reading, it broadcasts a warning packet in the neighbourhood that contains the suspect node's identification, the type of fault detected, the measurement of the suspect node and then triggers a waiting neighbour's reply timer. The neighbour, after receiving the warning packet, checks it by comparing the data in the received packet with its data. If the warning was wrong, the neighbour will reply to the message within the reply time. After the timer expires in the monitoring node, and if no message is received, the monitoring node sends a warning packet to the sink. If, during this assigned time, the monitoring node receives a reply, it will cancel its detection.

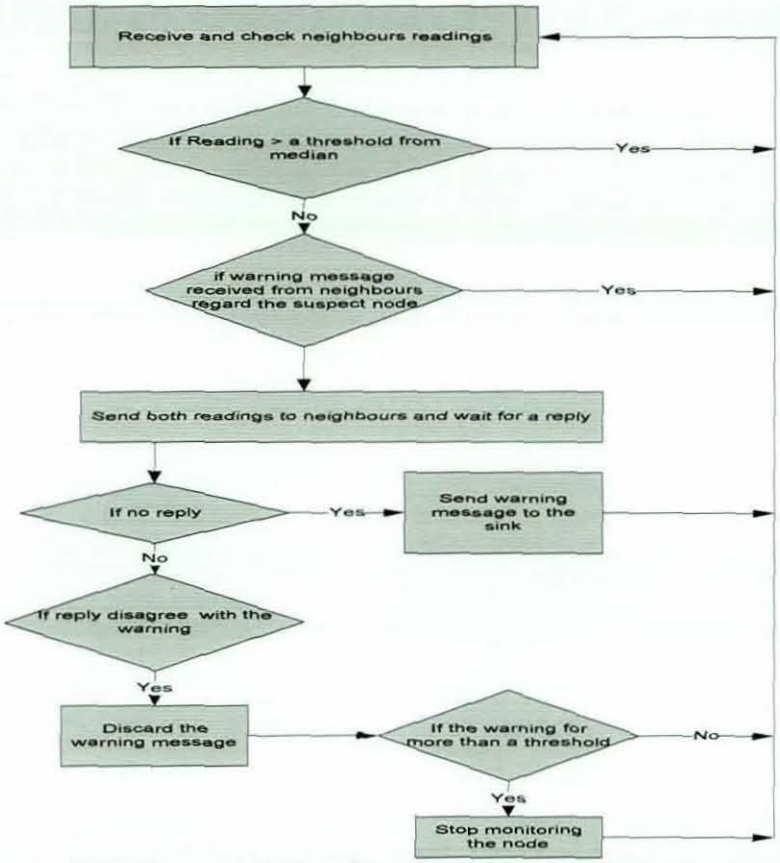

Figure 5-9. Proposed Algorithm for a Single Query Application

Implementing these modifications on the *MATLAB* code, and conducting simulation experiments of networks consisting of 500 to 1400 nodes randomly distributed over a 1000X1000 square metre area, each with a 50 metre transceiver range and using the Appendix A power consumption model, showed a good level of detection but with a linear increase in the number of exchange messages between neighbours, as shown in Figure 5-10.

Figure 5-11 illustrates the total number of send messages, the percentage of detections of deviated nodes, the total number of received messages, and the maximum amount of warning messages received by neighbours in the network when using a single query application and the modified algorithm with different percentages of 1400 nodes distributed randomly over a 1000 X 1000 square metre area in a 100-run simulation in *MATLAB*. The Figures 5-11.1 to 3 show that, as the number of deviated faulty nodes at the network increased to 50%, the algorithm's detection reduced by 6%. The sending of warning messages by the algorithm also increased as this reached 800 packets with

250,000 packets received in the network. In addition, Figure 5-11.4 shows that the maximum number of messages received at the node varies gradually because this depends on the position of the faulty deviated nodes.

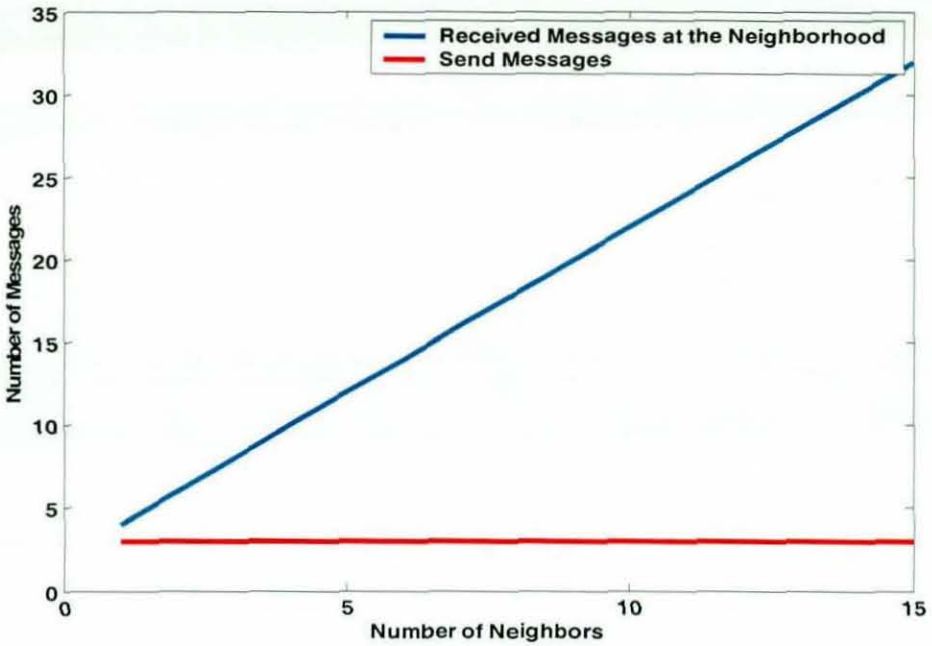

Figure 5-10. Number of Packet Exchanges between Neighbours in a Single Query Application

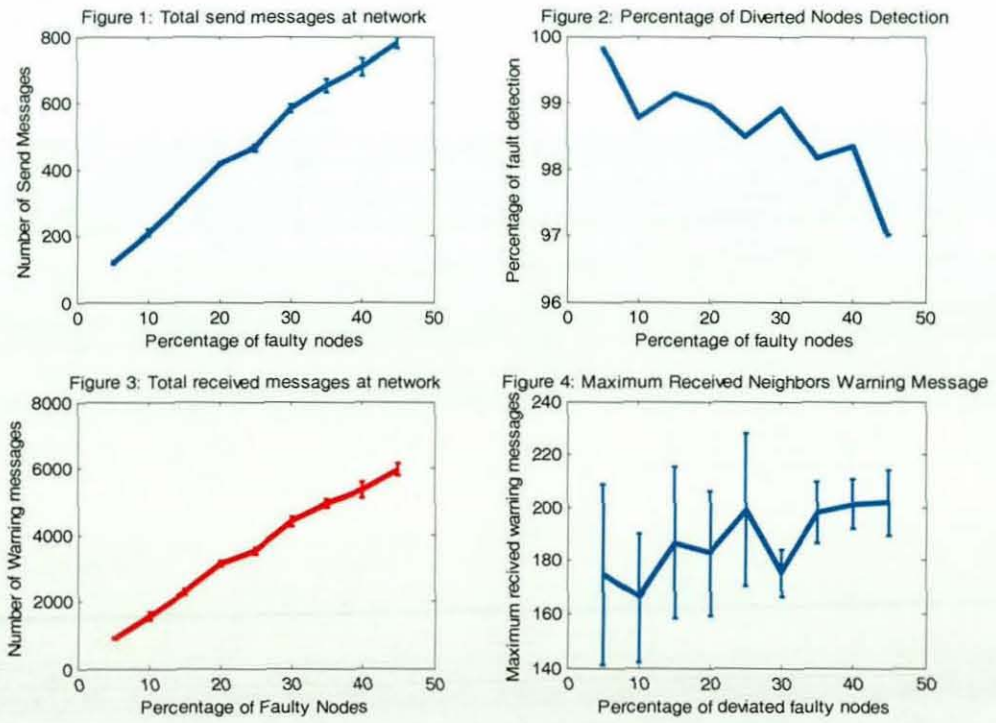

Figure 5-11. 1000 Nodes Inserted with Different Percentages of Fault Using a Single Query Method in a non-Aggregated Application

When these simulation experiments were repeated for 500 to 1000 deployed nodes in a 1000X1000 square metre area with 30% faulty deviated nodes, the algorithm's detection warning messages, sent and received, linearly increase as the number of nodes increases, as can be seen in Figures 5-12 A to C. However, the algorithm's detection increases exponentially with a low detection of deviation of 55% when 500 nodes were deployed; a sharp increase in detection (to around 80%) was noted when node deployments reached 600 with a linear increase after that until it reached 95% detection with the deployment of 1000 nodes. The maximum number messages received by the algorithm varied dramatically between 500 and 1000 deployments, as shown in Figure 5-12.D, due to different percentage of faulty nodes and their random distribution in the area.

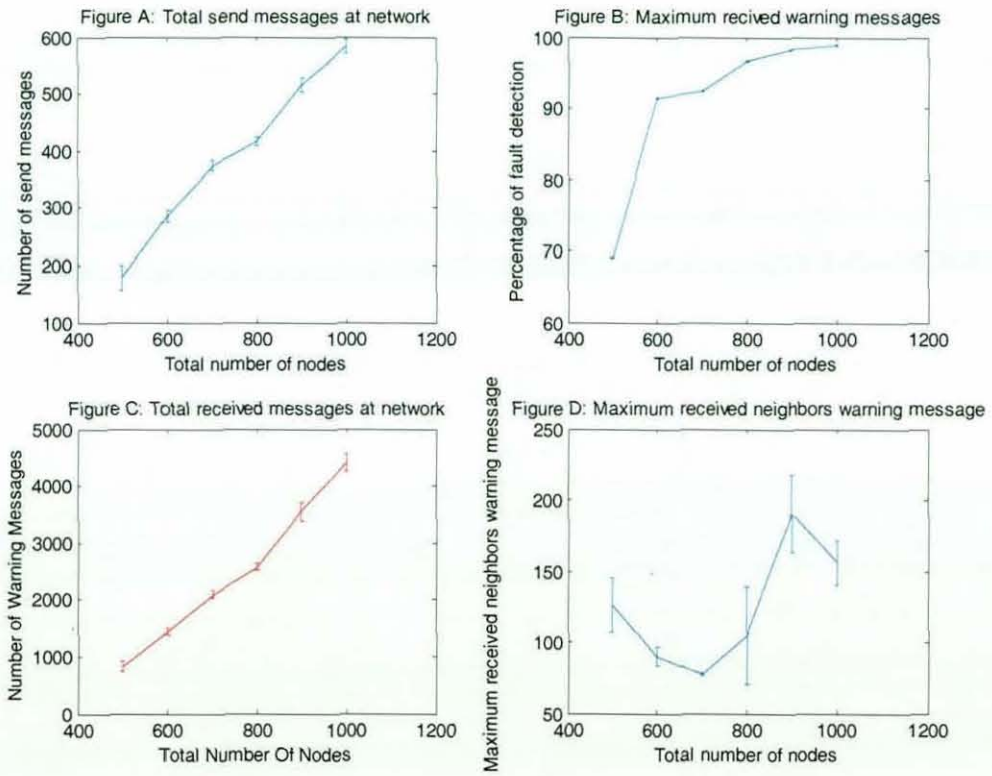

Figure 5-12. Single Query Algorithm Detection with Different Node Density Networks with 30% Deviated Faulty Nodes

5.3 Method 2 Algorithm Implementation

The second method uses the advantage of the negligible increase in energy consumption per packet as its size increases, as suggested by

Sankarasubramaniam *et al.* [114], although the probability of packet loss increases. The proposed algorithm was reconstructed such that it pushes down node measurement to low network level components and exchanges data with its neighours through updated routing table packets as is discussed in Chapter 8.

Testing the effect on the average node lifetime of sending 1000 packets in a network with an average of 16 neighbour nodes and power consumption model at Appendix A on MATLAB code, showed, however, that there was only a very small increase, as Figure 5-13 illustrates.

From this the algorithm was modified so that the application protocol pushed down its measurements to the communication layer which then inserted the latest measurement into the routing update exchange packet when it was sent. This modification helped in reducing the complexity of the algorithm code and made the algorithm independent of the application reporting rate so that it was no longer event driven, (as it will be discuss in Chapter 8). The algorithm, with this method, can be used for all applications without modifying its structure. It will detect the malfunction of nodes even if the assigned event threshold level is not satisfied.

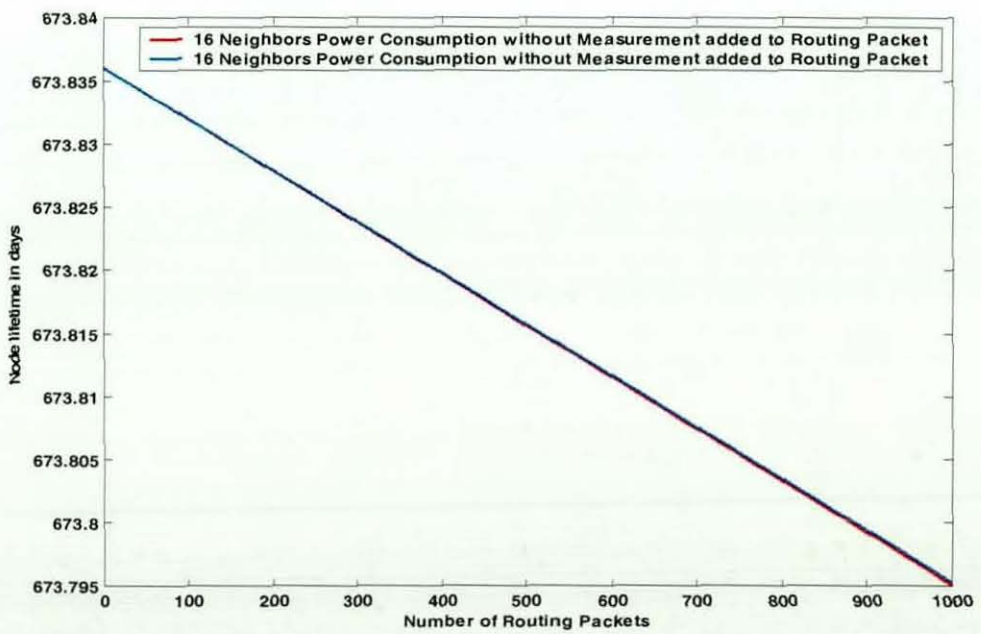

Figure 5-13. Network Lifetime Reduction with and without Inserting Measurements for the Application Layer in Update Routing Packet

The disadvantage of this method lies in its lower reporting rate from the communication layer. Because of this, the confidence of algorithm's detection will be lower than with the first method (i.e. around 33% less, depending on the communication layer's reporting rate). Therefore, to increase this confidence, either more monitoring time is required, or more validity tests on the collected data must be added, as discussed in Chapter 4. Moreover, in Method 2, losses have a greater effect on the implementation of warning messages in the network than in Method 1. This method is more effective, however, when the network is working with hybrid applications. (Experiments using such an implementation, together with the results of these tests, will be discussed in Chapter 8.)

5.4 Summary

This chapter discusses the different implementation strategies of the algorithm, together with its advantages, drawbacks and the modifications required to its structure.

Chapter 6 VMBA Algorithm Analysis and Performance Evaluation

6.1 Introduction

This chapter uses various scenarios to evaluate multiple aspects of the proposed algorithm. These include the algorithm's resource usage, its estimated value for the measured phenomenon, and its event detections. In addition, the chapter discusses the limitations that were detected during the simulation and the empirical experiments. The evaluations were carried out using several methods because of the *WSN's* complex functionality and to increase confidence in the results.

6.2 Algorithm Resource Usage

Resource utilisation is a very important metric in evaluating any new protocol or algorithm in a *WSN* due the constraints that directly affect a network's lifetime. As a result of this, the resource required by the *VMBA* was calculated using algorithm analysis techniques and simulation experiments.

6.2.1 Algorithm Analysis

Algorithm analysis was used to provide theoretical estimates of the algorithm's resource usage for such factors as memory, running time and the exchange of warning packets. This analysis was carried out for both best-case and worst-case scenarios. It gave an estimation of the proposed algorithm's resource usage formulae deduced from the algorithm's pseudo-code as shown in Appendix E, Table E-1. The best/worse cases in terms of resource usage were calculated from this. (This assumed that the best case was when all the node readings were within the threshold, and the worst case happened when half the nodes were deviated/faulty).

Table 6-1 shows the proposed algorithm's resource usage formulae that were calculated, as explained in [52]. These are plotted in Figures 6-1 and 6-2 to illustrate the proposed algorithm's memory usage, the running time required for its event analysis, and the amount of send/receive warning packets for the best and worse cases for different numbers of neighbour nodes.

	Formula	Best case	Worst case
Storage memory	$6n+14\sum_{k=1}^{n} k+18$	6n+18	$7n^2+13n+18$
Packet transmitting	$5\sum_{k=1}^{n} k$	0	$\frac{2n^2}{2}+\frac{5n}{2}$
Packet receiving	$\sum_{k=1}^{n-1} k\,(5\sum_{k=1}^{n-1} k)$	0	$5\left(\frac{n^2-n}{2}\right)^2$
Running time	$6n+25\sum_{k=1}^{n} k+10$	6n+10	$\frac{25n^2}{2}+\frac{37n}{2}+10$

Table 6-1. Algorithm Analysis Formulae in Best and Worst Cases[7]

These four metrics were used to analyse the results. The first metric presents the percentage of memory change, which is the ratio of the difference between the best and worst case scenarios in terms of memory usage to the worst case. This metric shows the impact of the number of neighbours on the algorithm's memory usage. The second metric is the change in the percentage of received messages, which is the ratio of the difference between the best to the worst in terms of the number of packets received and the worst case. This metric shows the impact of the number of neighbours and deviated neighbours on received packets. The percentage of the algorithm's running time is one metric that is used to evaluate the algorithm's length of process time for different numbers of neighbours. This is the ratio of the difference in the process times for the best and worst cases to the process times used in the worst case. This metric is used to show the percentage of change in the algorithm processing that increases loss; as discussed in [13]. The final metric to be used is the percentage of change in the number of packets sent by the algorithm, which is the ratio of the difference between the best and worst cases in terms of sent packets to the worst case. This metric shows the impact of the number of neighbours and deviated neighbours on the sent packets.

[7] n is the total number of nodes appearing in the algorithm time interval calculation due to losses.

Figures 6-1 and 6-2 show that, as the number of neighbour nodes increase, the memory size required for monitoring neighbourhood nodes also increases, with a change between the best and the worst cases reaching 90% with 16 neighbour nodes. This means that there will be a large increase in energy consumption and in data processing and retrieval time if the data are stored in Electrically Erasable Programmable Read-only Memory (*EEPROM*). This is because *EEPROM* has a relatively high read and write power consumption and needs a relatively long time for data retrieval. To reduce this energy consumption and save time, the algorithm was designed such that received neighbour readings analyses were stored in the node *RAM*. However, a *RAM* is considered to be a very tight resource in current node platforms. (This is discussed in Chapter 8.) The main problem with this solution is that data are lost when the node is initializing.

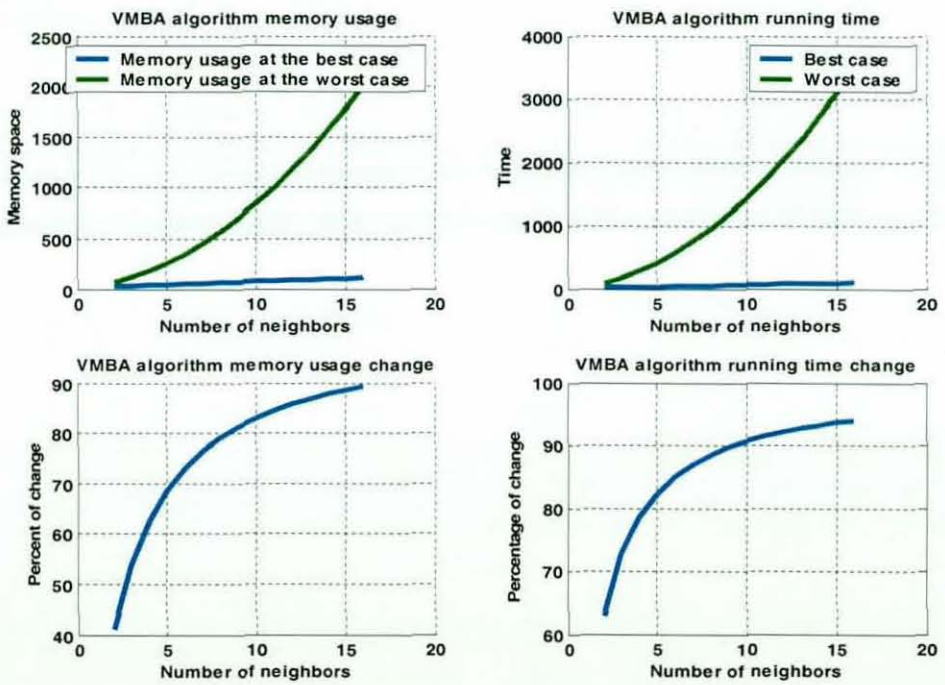

Figure 6-1. Algorithm Memory Usage and Running Time of the Proposed Algorithm at Different Neighbour Densities

Figure 6-1 also shows an exponential increase in the algorithm's running time as the number of neighbour nodes increases from 2 to 16, with a difference between the best and the worst cases reaching 94%. Therefore, as the number of neighbours increases, the processing time also increases. This running time is very important in terms of practical implementation. If the

hierarchical timing arrangement of modules and functions is not followed in the network application flow process, the complete application may collapse (i.e. this detected in empirical experiments and cause the application to stop completely). This drawback was solved by using a *'post'* task *nesC* command to schedule and manage the task processing. In addition, the functions of the proposed algorithm were integrated into the flow process of the network application. Although this solved the problem, the empirical experiments showed some of the algorithm's warnings were lost (as discussed in Chapter 8).

However, the experiments showed that this exponentially increasing running time and memory usage varied due to the heavy packet losses that *WSNs* face. This causes a variation in the memory size required for storage at each event, which, in turn, leads to a variation in the algorithm's power consumption and resource usage. For example, Figures 6-1 and 6-2 show that, if the number of received readings change between 2 and 6, the required memory size increases by 30% while the algorithm's running time increases by 20%. To reduce the effect of this exponential increase (i.e. in memory usage and running time), a conditional statement was added to force the algorithm's event analysis to start up only if a deviation was detected in neighbour node operation.

Figure 6-2 shows that the transmission percentages of warning messages stay constant with increases as the number of neighbour nodes, while the number of received packets in the network increases exponentially. This exponential increase in the number of received packets causes more energy consumption in nodes especially if these were not using power-saving techniques. In addition, this may cause a high level of traffic in the network that will, in turn, increase congestion and packet losses. To reduce this effect, the proposed algorithm was designed to include a packet exchange module. This module stores all warning messages sent in the neighbourhood and checks the storage memory contents before a sending warning packet; as explained in Chapter 4. In addition, this module stops reporting the same fault warning after a predefined time. Although these two solutions helped in

reducing power consumption and reducing network traffic, they increased the probability of losing warning message especially in multi-hop exchanges. This is discussed in Chapter 8.

Finally, Figure 6-2 shows that the number of warning messages sent differs by 100% between the best and the worst cases. This is because the proposed algorithm does not send any message unless there is event detection.

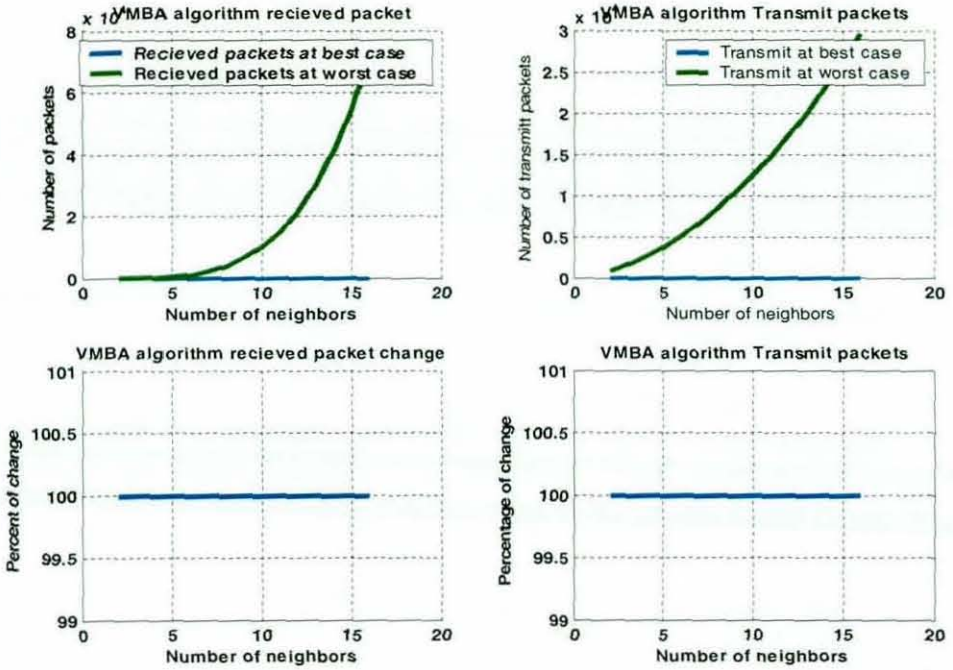

Figure 6-2. Transmit and Received Packet Exchange of the Proposed Algorithm at Different Neighbour Densities

A comparison of the median analysis used in module 2 of the algorithm, with three simple analyses selected from statistical, fusion and analytical redundancy methods, was conducted. This was done to evaluate the median method that was used. The methods that were selected depend on their low level of resource usage and their high efficiency in estimating phenomenon values.

A statistical redundancy method was selected as the average method for making comparisons with the median. The estimated running time and the memory growth of the average method is in the range of $O(n)$ while the median running time growth is $O(n^2)$ if the divide-and-conquer method is not

used; if this method is utilised, it is $n\log(n)$ [52]. The rate of growth of the median memory, however, is in the range of *O(2n)*. Although the amount of memory used by the median method is higher, the average analysis breaking point, (i.e. the point at which the change affects the calculation), is 0, as explained in [109]. This means that any deviated node will affect the estimated neighbourhood value in an average method while the median method is only affected if 50% of the readings are deviated, as discussed in [109].

If the fusion method employed by Unter der [53] is compared with median analysis, the growth in running time is in the range of *O(n)* while the memory storage depends on the conversion table size plus *O(n)*. The main problem with this method is that its storage capacity depends on the accuracy of its conversion table, while the table contents depend on the sensor that is used; this is not required by the median method.

Finally, if the adaptive prediction base employed by Santini in [54] is used as a simple analytical method, it has a running time and a memory growth of *O(k*n)*, where k stands for the historical measurements of the phenomenon that are used. The main problem with this method is that its accuracy depends on these historical measurements. Moreover, it requires a learning time and the method performance affected by high packet loss.

6.2.2 Energy Consumption

Power consumption is one of the most important performance metrics in *WSNs* due to heir energy limitation and the difficulty of changing the power supply after they have been deployed. The energy consumption of the proposed algorithm was tested using a *MATLAB* simulation and the results were verified by using *'PowerTossim'* simulation. Using the power consumption model illustrated in Appendix A, Table A-1, the algorithm's energy consumption was tested by simulating a set of 1000 nodes randomly

distributed over a 1000X 1000 square metre area. These sensors had a 50 metre transceiver range and a Model 2 Mote duty cycle; i.e. 5% to 95%[8].

The percentage of additional energy consumption was the metric used for evaluating the result. This is the ratio of the difference between the power consumption with and without using the algorithm compared to the energy consumption when the algorithm is not used. This metric illustrates the percentage of extra energy used by the algorithm.

The experiments showed that there is a small difference in the amount of energy consumption between implementing or not implementing the algorithm, as shown in Figures 6-3 and 6-4. This small consumption came from transmitting warning messages when the algorithm detected faults and algorithm events computation.

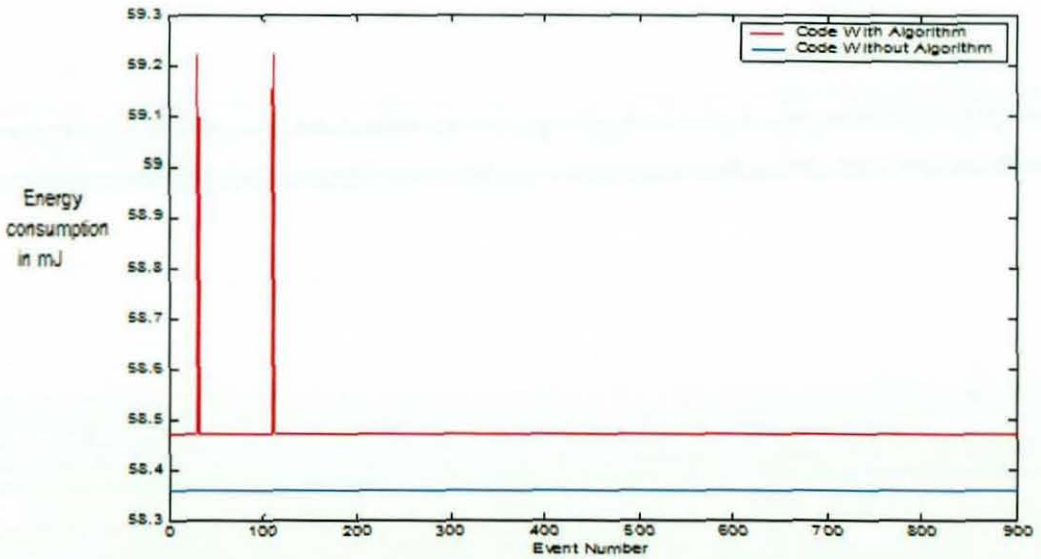

Figure 6-3. Energy Consumption with and without the Algorithm

The same experiment was then repeated using *'PowerTossim'* [47], [115] with 60 virtual durations, a 2-second reporting rate, 0x09 transmission power (i.e. -10 dBm), and a Model 1 duty cycle (i.e. 1% to 99%). The comparison between application codes with and without the proposed algorithm shows a slight increase in processing and total energy consumption (as shown in Figure 6-5).

[8] Please note that the affect of change over time between sleeping and active nodes was not considered in this model.

The difference between the energy consumption in the two *CPUs* was calculated, as shown in Figure 6-6, and it was found that the average difference was 4.4 ± 1.5, with maximum of 5.9 mJ; this is around 0.8% of *CPU* energy consumption without using the algorithm. This consumption varied with the number of neighbours and was at a maximum when the number of nodes in the neighbourhood was fewer than four, as shown in the figure. This is because of the increase in node wakeup time as the number of neighbours increases. This makes the *CPU* power that is consumed almost negligible when compared to other processor operations for the same period. Moreover, if the algorithm does not detect a deviation in the operation its process will not begin.

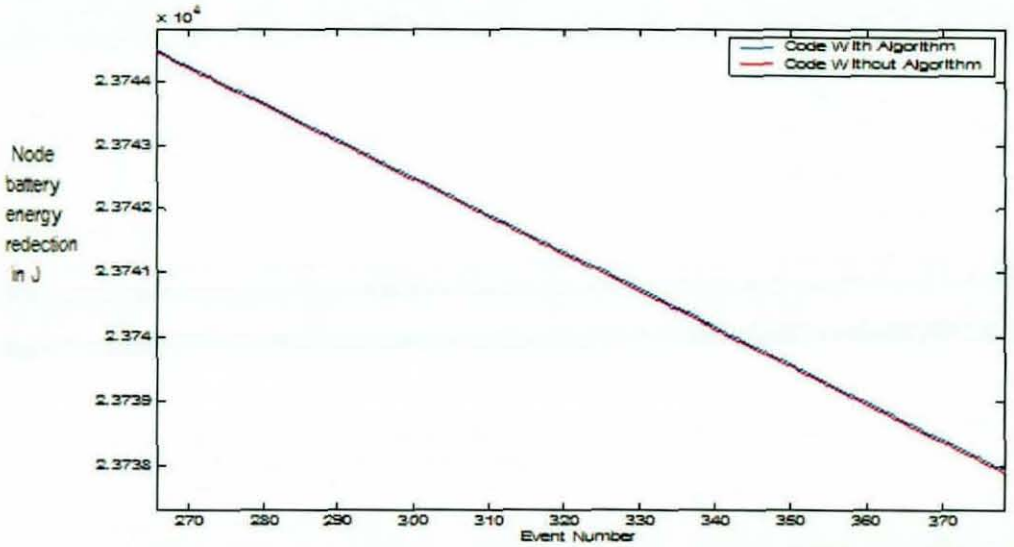

Figure 6-4. Node Battery Energy Consumption with and without the Algorithm

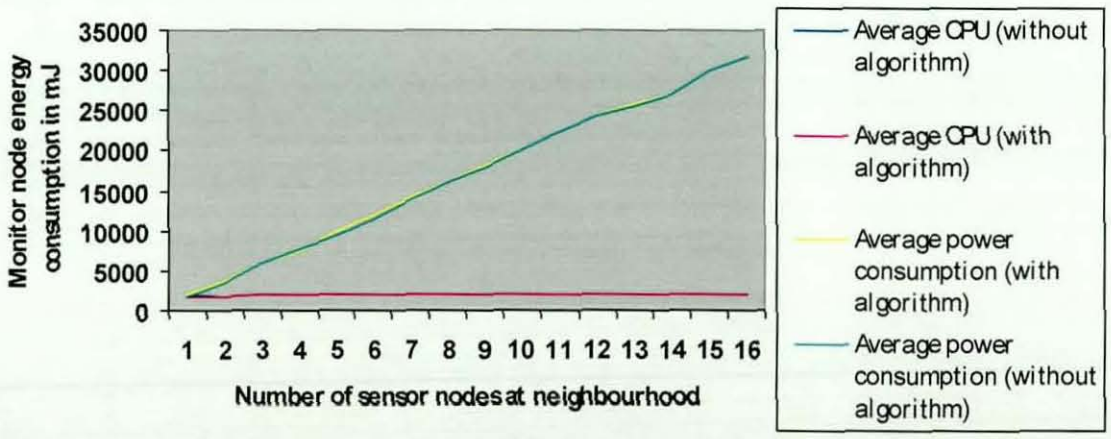

Figure 6-5. CPU and Total Energy Consumption with and without the Algorithm

In order to calculate the effect of the proposed algorithm on the network lifetime in terms of the best and worst cases, the number of warning packets released in the two cases was calculated; using the same calculation method in [87]. The algorithm releases a message when it detects an event (i.e. node malfunction, a dead node, neighbourhood malfunction, accuracy of the neighbourhood collected data, coverage, or connectivity warning messages) or when it does not agree with a received neighbour warning packet. Appendix E, Table E.2 shows the calculation of each event detected by the algorithm in the best and the worst cases. The metric used to evaluate the result was the reduction in the nodes' lifetime when the algorithm was used. This metric shows the effect of the algorithm's work on the lifetime of nodes.

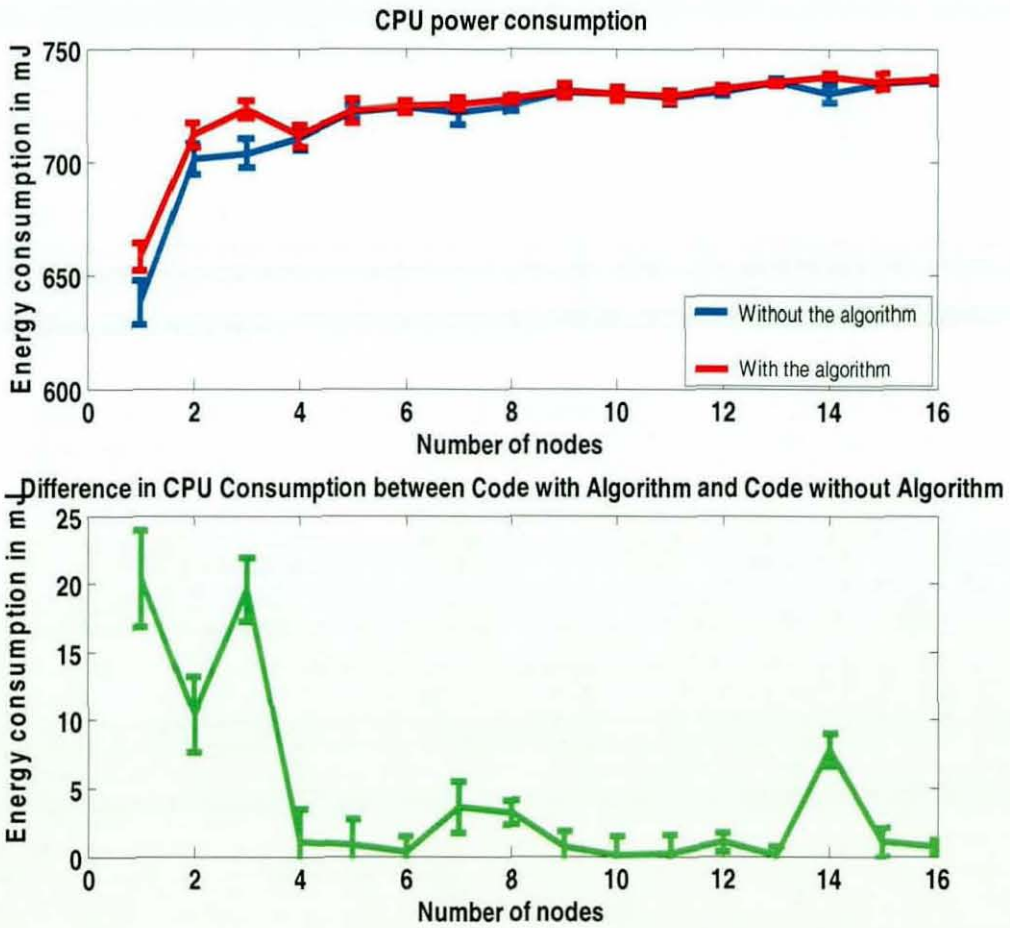

Figure 6-6. CPU Energy Consumption between Codes with and without the Algorithm

Figure 6-7 illustrates the reduction in node lifetime caused by the algorithm packets that are released and exchanged in a worst case, with different

average numbers of neighbour nodes. As the figure shows, the maximum reduction in node lifetime, with 16 neighbour nodes, is around 0.03 days; this equals 0.01% of the total lifetime[9]. This low reduction was a result of the algorithm's module 4 that controls warning messages.

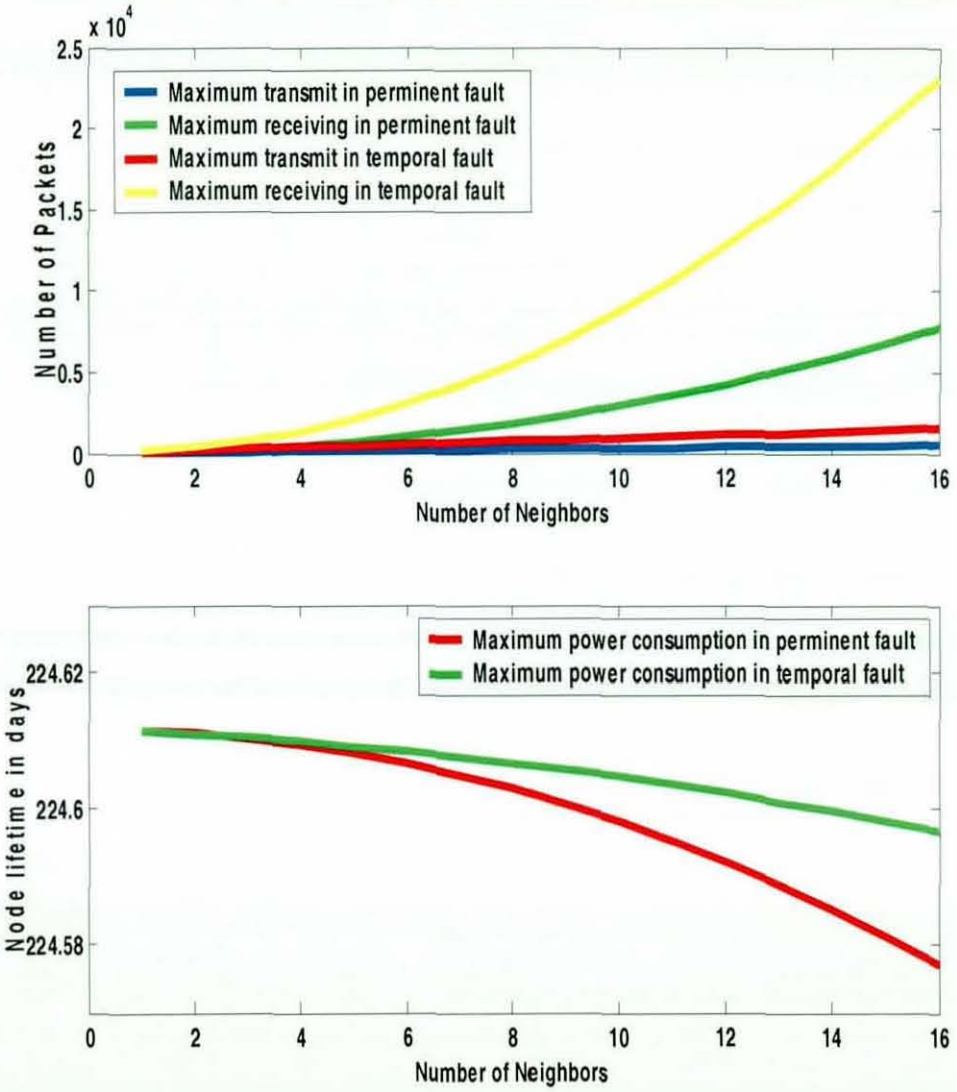

Figure 6-7. Maximum Packet Transmissions and the Expected Node Lifetime with Temporary and Permanent Neighbour Faults

To evaluate the energy consumption of the warning message exchange of the proposed algorithm and the multi-hop routing consumption, *MATLAB* code was used to conduct simulations. The simulation scenarios were set so that there were 1000 nodes with a 50 metre sensing range, randomly distributed

[9] The calculation was carried out using the power model shown in Appendix B, Table B.2 on the '*TinyOS Surge*' application with Mica2

over a 1000X1000 square metre area (i.e. giving an average of 7 neighbours). Each node reported its packets to a sink through a combination of a distributed election leader and distance-vector routing algorithms (as described in [23]). The algorithm's consumption was compared with two commonly used *WSN* techniques, (central and aggregated, as described in [23]). When the central method is used, all nodes send periodic reports, (that is, every 10 samples in the simulation), and with aggregation scenarios the cluster head node aggregates and sends the result every 10 samples. The data used in the simulations were as follows: 40 dead nodes, 10% random deviated data, 30% packet losses, and 100 permanently deviated nodes.

The outcome from these simulations for each method is summarised in Table 6-2. The table shows that the proposed algorithm consumes only 10% compared to the centralised method and 18% compared to the aggregation method. Much of this reduction is because warning messages are sent when required; also a stop reporting method was used in module 4. Furthermore, the results showed that the main consumption for the three methods was due to the multi-hop routing consumption. The experiments illustrated that the multi-hop consumption necessary for sending the required algorithm packet was, on average, between 87%-95% of the total power (This work has been left for a future extension of the scope of this study).

	Central		Aggregation		Algorithm	
	Multi-hop	**Packet creation**	**Multi-hop**	**Packet creation**	**Multi-hop**	**Packet creation**
Maximum	$1.2X10^3$	6.6	636.7	6.6	71	1.1
Minimum	0	1.1	0	1.1	0	0
Mean	42.9	2.3	23.9	2.1	4.7	0.3
Standard deviation	100	1.1	56.5	0.8	8.3	0.2
Total	$4.3X10^4$	$2.4X10^3$	$2.4X10^4$	$2.1X10^3$	$4.6X10^3$	245.9
Total network consumption	$45.4X10^3$		$26.1X10^3$		$4.8X10^3$	

Table 6-2. Energy Consumption of the Packet Exchange in mJ for Central, Aggregated and the Proposed Algorithm Packets Exchange

6.3 The Algorithm Estimation Value for Phenomenon Measurement

Median calculation is very important because the algorithm's analysis depends on it. As a result of this, the accuracy of the calculated median in predicting the value of phenomena was tested using robust algorithms that estimate such values using model, linear prediction and fusion techniques. This is due to the lack of availability of ground truth for the measured phenomenon values[10].

The metric used to analyse these results was relative error. This is the ratio of the absolute difference between the value calculated by the median and that estimated by the method compared with the value estimated by the method at a particular time interval. This metric computes the accuracy level of the calculated median value using more complex methods and with methods that use more resources.

The Time Triggered system protocol (*TTP/A*) [53] uses an Weighted Moving Average Fusion (*WMAV Fusion*) algorithm with a confidence between 0 (i.e. the lowest level of confidence) and maximum confidence (i.e. the highest level). This confidence can be interpreted as an estimator of statistical variance V[s] and of measurement error, with variance being the second moment of arbitrary probability density function. In this algorithm, the assumption was made that the correct measurement of variance in the phenomenon is closer to 0, which corresponds to the maximum level of confidence. In the worst case, a sensor delivers for the measurement a random value within its range. The worst-case variance is calculated as the variance of a uniformly distributed random function between limits a and b: $V[s]=\frac{(b-a)^2}{12}$ where a and b are the minimum and maximum values of the expected uniformly distributed random function. The measurement values are fused by using a weighted average with the reciprocal variance values as

[10] The ground truth is the actual measurement of the phenomenon.

weight: $\bar{x} = \frac{\sum_{i=1}^{n} x_i \frac{1}{V[s_i]}}{\sum_{i=1}^{n} \frac{1}{V[s_i]}}$ where n is the number of input observations, x_i is the measurement and V $[s_i]$ is the estimated variance from the measurement. The statistical variance of the observation output is:

$$V[s_i] = \frac{1}{\sum_{i=1}^{n} \frac{1}{V[s_i]}} \qquad (5.1)$$

Figures 6-8 and 6-9 show comparisons between the median and the estimated neighbourhood values using the *TTP/A WMAV* Fusion method. The experiments used an Intel Lab experiment data set [57] with both lossless and 65% loss. The *TTP/A WMAV Fusion* method was also used to evaluate values for temperature measurements, as shown in Appendix E, Table E-3. The figures show that the maximum detected error between the two methods is around 2% with 65% packet losses. This proves the high level of reliability of the median value used in the proposed algorithm for approximate measuring of the estimated phenomenon value.

The same experiment using the same data set was repeated for linear prediction [55] and Gaussian correlation algorithms [13]. Figures 6-10 to 6-14 show comparisons between the second order linear prediction technique, Gaussian correlation, *TTP/A WMAV Fusion* [53] and the median at both 65% loss and lossless mediums. These figures show that prediction and tracking techniques predict more stable phenomenon values than the median used by the proposed algorithm at a high loss medium; especially in Figure 6-11. This stability came with a need for greater processing complexity, more memory space for historical values, and a greater level of code complexity, as discussed in [55]. In addition, these experiments showed that the maximum relative error between these methods and the median was around 4%, while the maximum mean squared error was around 1.8 degrees centigrade, as shown in Table 6-3. These error levels are within accepted accuracy levels for the designed sensor characteristics, as shown in [88], [94].

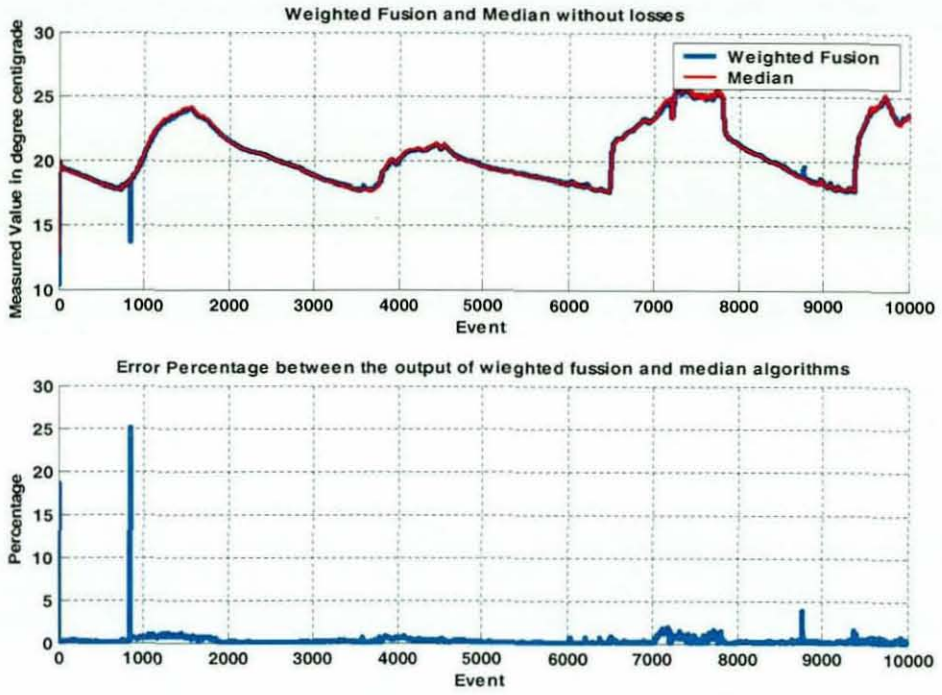

Figure 6-8. Percentage of Error of Detection between the Weighted Moving Average Fusion Algorithm (TTP/A WMAV Fusion) and the Median Algorithm in a Loss Less Medium

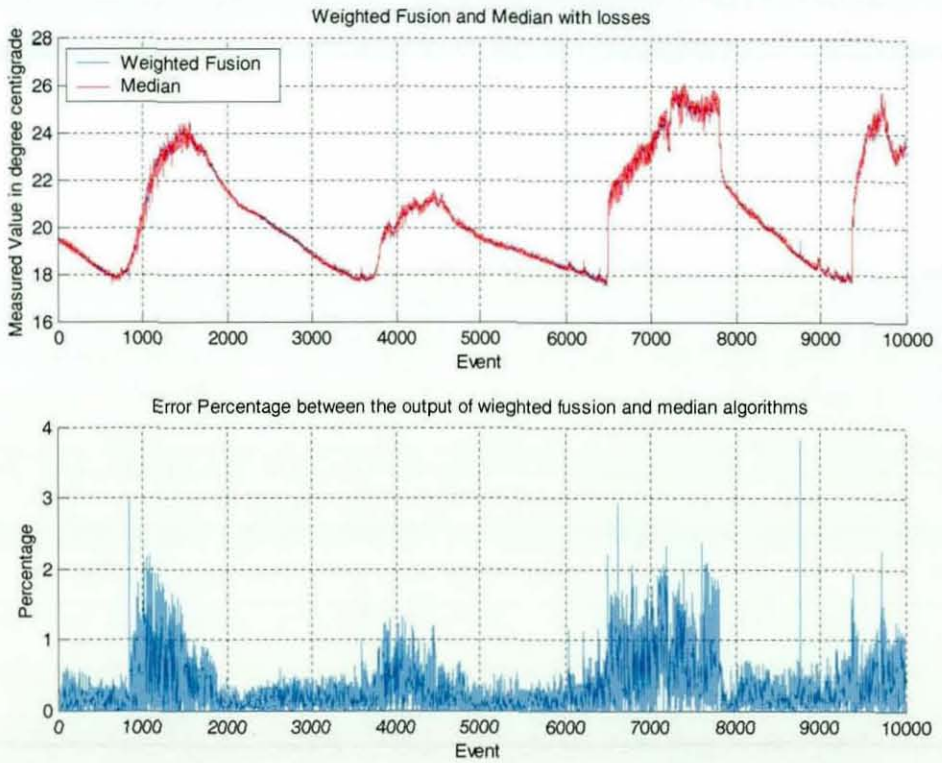

Figure 6-9. Percentage of Error of Detection between the Weighted Moving Average Fusion Algorithm (TTP/A WMAV Fusion) and the Median Algorithm with 65% Loss Medium

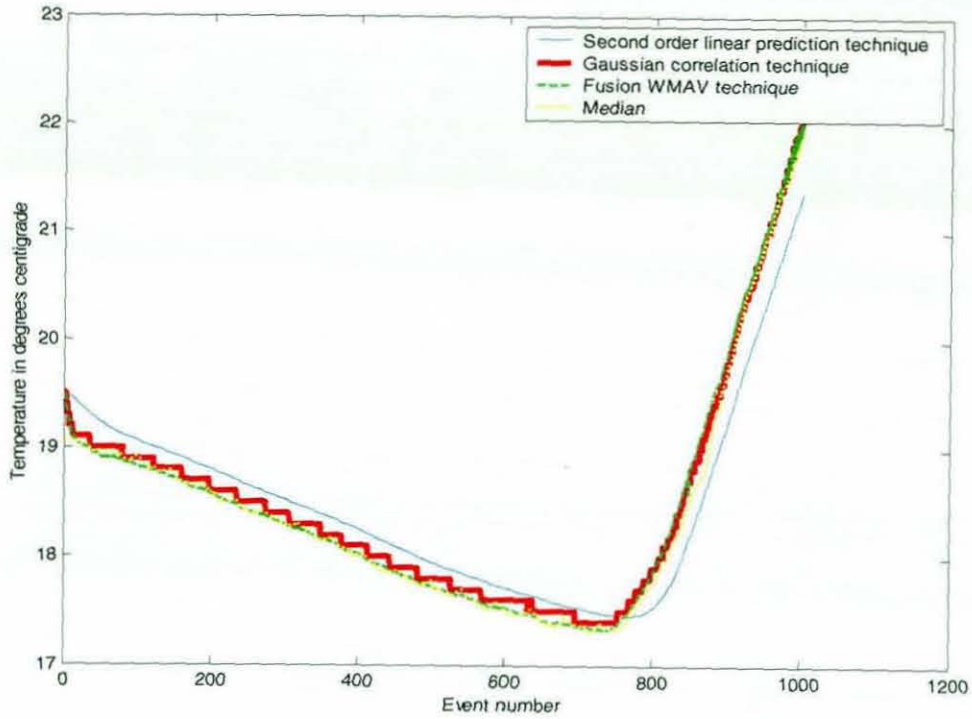

Figure 6-10. Value Calculation of Median, Second Order Linear Prediction, WMAV Fusion, and Gaussian Correlation without Losses

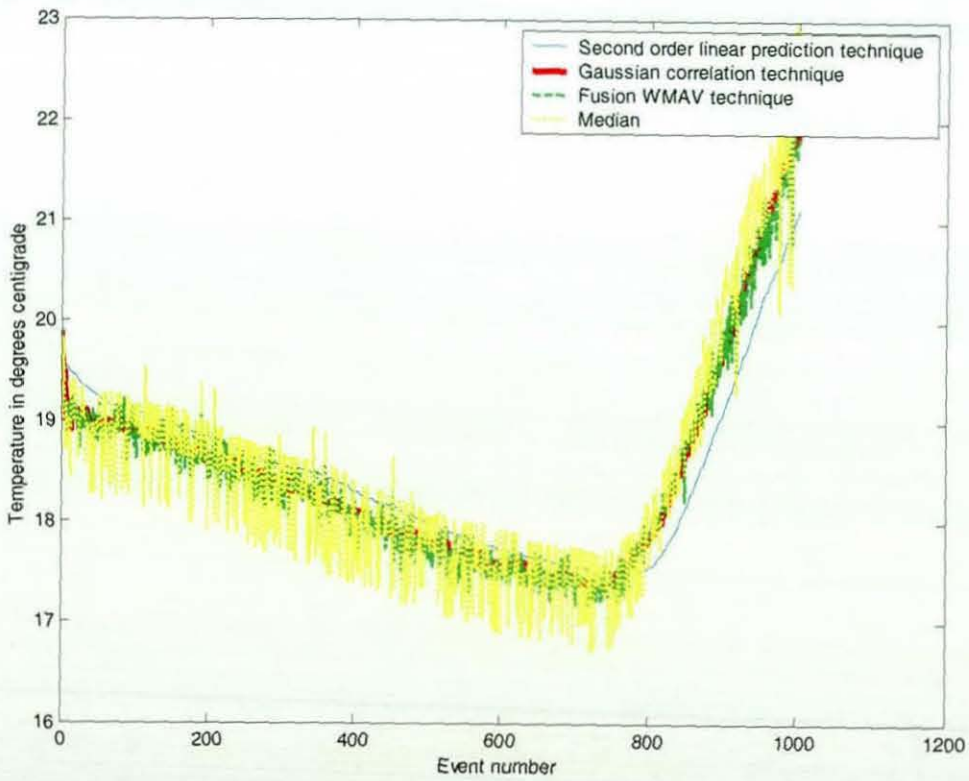

Figure 6-11. Value Calculation of Median, Second Order Linear Prediction, WMAV Fusion, and Gaussian Correlation with 65% Losses: Intel Lab data set

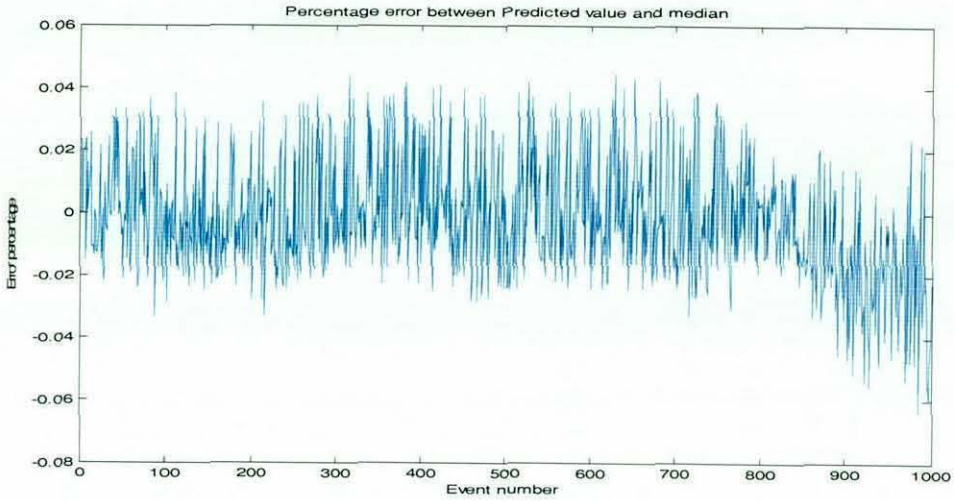

Figure 6-12. Differences between Values Calculated by Median and Second Order Linear Predicted Approaches

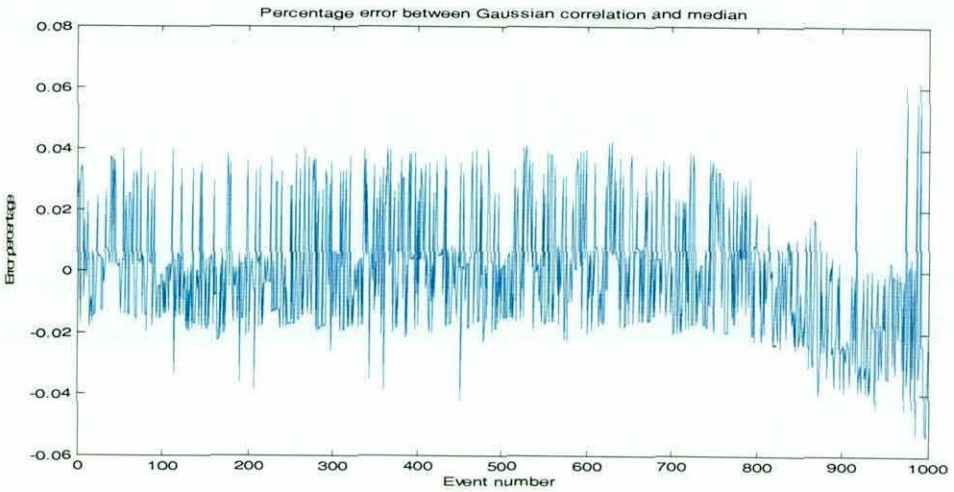

Figure 6-13. Differences between Values Calculated by Median and Gaussian Correlation Approaches[62]

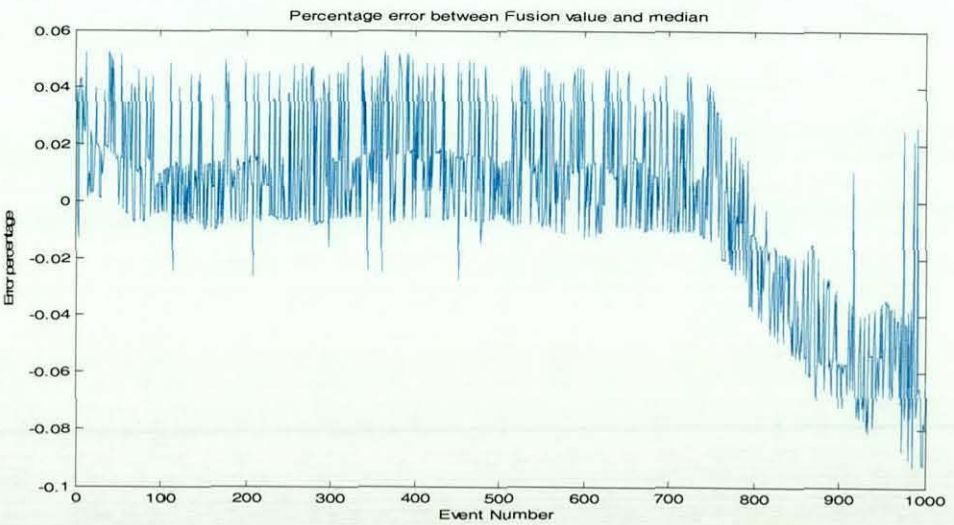

Figure 6-14. Differences between Values Calculated by Median and WMAV Fusion Approaches[66]

Loss	Predicted method		Moving Weighted Average Fusion method		Gaussian method	
	Mean squared error	Absolute error	Mean squared error	Absolute error	Mean squared error	Absolute error
0%	0.002	0.045	0.02	0.14	0.02	0.14
38%	0.5	0.7	0.21	0.51	0.19	0.43
65%	1.8	1.3	0.1	0.4	0.18	0.43

Table 6-3. The Absolute and Mean Square Error for Predicted, Moving Weighted Average Fusion, and Gaussian Methods and the Proposed Algorithm in Degrees Centigrade for Different Losses (in C^o)

6.4 Algorithm Detection

The goal of the algorithm is to detect the data that have a high impact on the accuracy of the collected data and the network's functionality; it is not intended to detect all deviated data. As a result of this, the operation of the proposed algorithm can be validated by three methods. The first is carried out using a simple analytical model. This is done by testing the probability of the algorithm detecting, or falsely detecting, deviations depending on threshold values. The metrics used for the analysis of these results were the probability of the algorithm's detection, and the percentage of detected faults. The first is the ratio of detected deviations compared to the total number of deviated nodes in different scenarios of threshold settings and different numbers of neighbours. This metric illustrates the change in the algorithm's detection with a range of threshold values and numbers of neighbours. The second metric, on the other hand, is the ratio of the number of detected faults compared to the total number of actual faults for different threshold values and different numbers of neighbours.

The second method compares the algorithm's detection with statistical analyses, such as the multi-factor analysis of variance of data, (i.e. using the Box-Whisker method [50] and the Bayesian method [23]). The metric used for analysing this result was the percentage of detected faults, which is the ratio of the number of faults detected by the algorithm compared to the detection

achieved using the Box-Whisker or Bayesian methods. This shows the robustness of the proposed algorithm in detecting outliers.

Finally, simulation experiments were conducted to test the detection of faults, in terms of both the algorithm's positive and negative false detections, using simulated data with different fault percentages. The metrics used to analyse these results were both positive and negative false detections. The first is the ratio of healthy nodes detected by the algorithm as deviated, as opposed to the total number of detections. The second is the total number of undetected deviated nodes compared to the total number of deviated nodes in the network. Both metrics illustrate errors in the algorithm's detection for different scenarios.

In order to test the consistency of the results obtained using different methods, a comparison of the algorithm's detection using the three methods was carried out for 10 node configurations.

6.4.1 Analytical Calculation of the Detection performance

In order to calculate the probability of the proposed algorithm's correct detection, the Bayesian method was used. I.e.

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B \mid A)p(A) + p(B \mid A^c)p(A^c)} \qquad (6.2).$$

This is because the detection depends on observing the number of operational values within the assigned threshold. The assumptions made for this model are:

- Neighbour nodes are those within sensing range.
- Each node in the neighbourhood has equal weight regarding its measurements in the algorithm's analysis.
- The phenomenon characteristics are spread out geographically over multiple neighbouring sensors so that faulty deviations can be distinguished from actual changes in the phenomenon by checking the correlation of neighbour readings.
- Faulty deviations of neighbour sensors are uncorrelated.

- To distinguish correlated nodes, the assigned threshold value depends on the accuracy that is required between two neighbour nodes. This threshold value has a false probability (i.e. p) equal to the ratio of the difference between it and the expected change in the phenomenon at the end of the sensing range, compared to the expected change in the monitoring at the end of the sensing range.
- Measurements that are larger than the calculated median value, plus the threshold value, are considered as deviations, while measurements within this interval are considered to be correlated. So, if the number of nodes at a neighbourhood is N and if r of these are within the assigned threshold, then the probability that nodes are correlated is $\frac{r}{N}$.

From this, the probability of the algorithm detecting deviated measurements is the probability of a deviated measurement being obtained when there are r of N neighbourhood measurements within the predefined threshold value. This is:

$$p(measurement\ is\ deviated \mid number\ of\ measurements\ from\ neighborhood\ nodes\ within\ the\ threshold)$$

Let

$$A = p(number\ of\ measurements\ from\ neighbour\ hoodes\ within\ the\ threshold \mid measurement\ is\ deviated) = \frac{r}{N}$$

$$B = p(measurement\ is\ deviated) = 1 - p$$

Using formula 6.2:

$$P(\text{detected measurement by the algorithm is deviated}) = \frac{\frac{r}{N}(1-p)}{\frac{r}{N}(1-p) + \frac{N-r}{N}p} \qquad (6.3)$$

Figure 6-15 illustrates the correct detection probability versus the threshold false probability: (the neighbourhood is with 10 nodes). This figure shows that, as the probability of false threshold detection increases, the probability of correct detection decreases. This decrease occurred at varying speeds, depending on the number of correlated measurements in that neighbourhood. For example, at a false assigned threshold probability of 0.3, if 3 neighbours were within the assigned threshold value, the probability of the correct detection was 0.56. However, if the number of correlated nodes increased to

9, the probability of detection increased to 0.96. In addition, the figure shows that the speed of detection changes between these two numbers of correlated nodes, as discussed above, for different levels of false threshold probability.

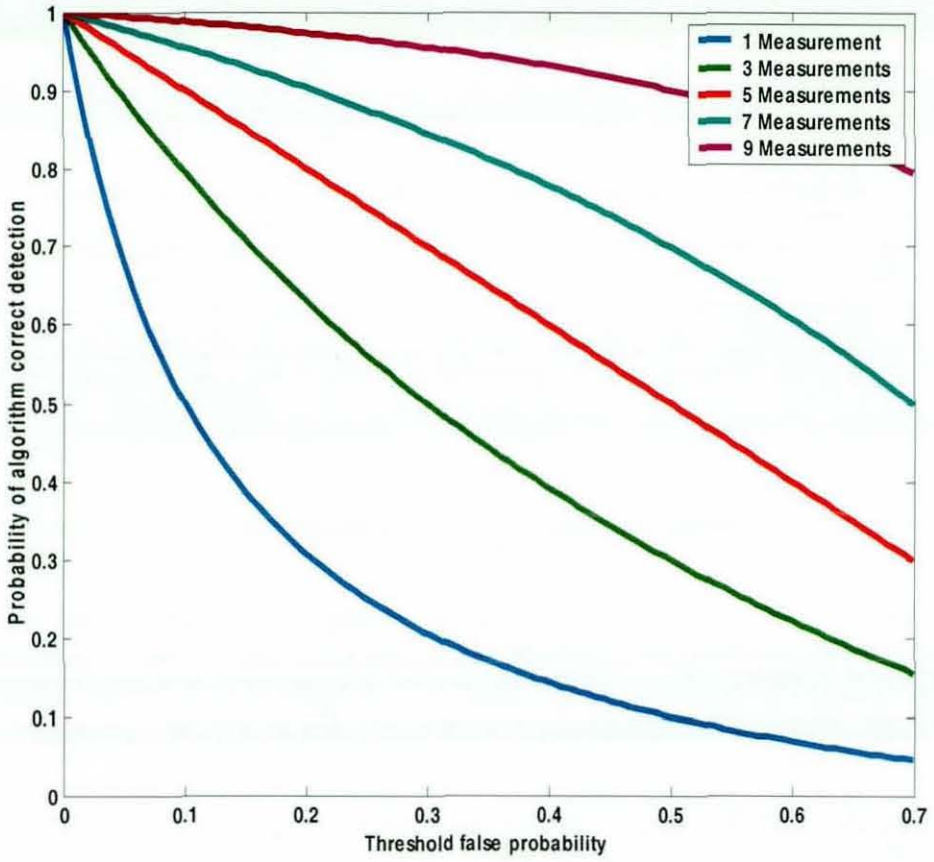

Figure 6-15. The Relation between the Probability of Correct Detection as Opposed to the Threshold's False Probability

To calculate the probability that the monitor node will detect, *'k'*, i.e. the faulty deviated nodes in the neighbourhood, the same Bayesian formula can be used: formula (6.2). This is done by calculating the probability that the algorithm will detect *'k'* deviations when there are exactly *'k'* deviated nodes in the neighbourhood, as in:

$$p(algorithm\ detects\ k\ deviated\ nodes | there\ are\ exactly\ k\ deviated\ nodes) \qquad (6.4)$$

The calculation for formula (6.4) requires the probability that there are exactly *'k'* detections by the algorithm in the N neighbourhood measurements. This can be deduced by using the binomial distribution probability because the

detection of a deviation can be either because it detects deviated or healthy nodes outside the assigned threshold value.

$$p(\textit{exactly k neighbors readings in neighborhood are deviated}) = \binom{N}{k} C.D \quad (6.5)$$

Where

$C = p(\text{measurement is deviated})^k = (1-p)^k$

$D = p(\text{measurement is not deviated})^{N-k} = p^{(N-k)}$

Using formula (6.5)

$$= \binom{N}{r} (1-p)^k p^{(N-k)} \qquad (6.6)$$

Also, the calculation for formula (6.4) requires the probability that the algorithm will detect *'k'* faults, as explained in formula (6.3). Using formulae (6.3) and (6.6) to calculate formula (6.4), results in:

$$\frac{\binom{N}{N-k}(1-p)^{N-k} p^k \dfrac{\frac{k}{N}(1-p)}{\frac{k}{N}(1-p)+\frac{N-k}{N}p}}{\binom{N}{N-k}(1-p)^{N-k} p^k \dfrac{\frac{k}{N}(1-p)}{\frac{k}{N}(1-p)+\frac{N-k}{N}p} + \binom{N}{N-k}(p)^{N-k}(1-p)^k \dfrac{\frac{k}{N}p}{\frac{k}{N}p+\frac{N-k}{N}(1-p)}} \quad (6.6)$$

Plotting Formula (6.6), as shown in Figure 6-16, illustrates the percentage of change in detection with increases in the number of deviated nodes. The figure shows that, as the number of deviated nodes increases, the detection reduces by a certain percentage, depending on the level of the false probability in the threshold. Furthermore, the figure shows that, as the false probability tin the hreshold decreases (i.e. the threshold value becomes larger), the probability of detection increases. For example, if the number of deviated nodes is 4, the detection percentage increases from 67% with a false probability threshold of 0.3; it then increases to 88% with a false probability threshold of 0.1. However, this also increases the value of the assigned threshold and, in such cases, the algorithm may not detect deviations that occur due to the neighbourhood coverage of nodes.

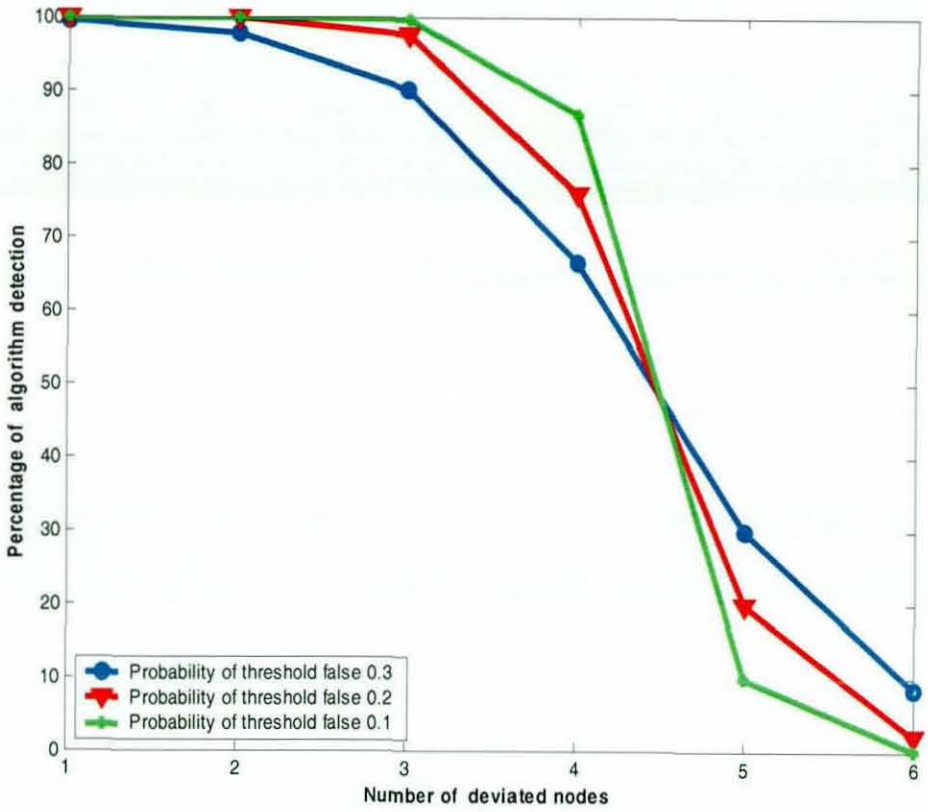

Figure 6-16. The Relationship between the Detection and the Threshold *False*

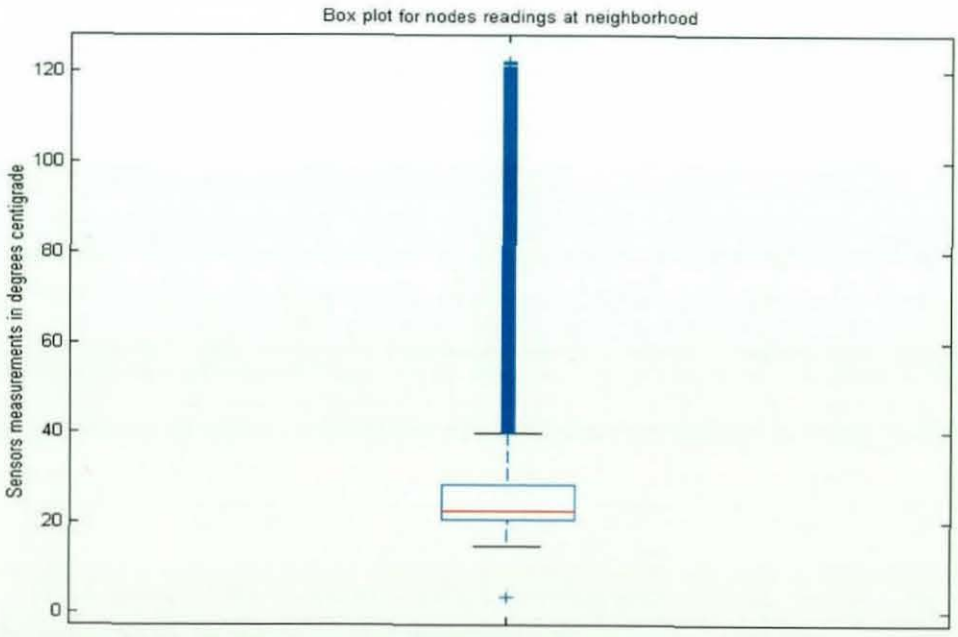

Figure 6-17. Statistical Method to plot Outlier Locations using Lab Experiment Measurements

6.4.2 Validating the Detection Using Statistical Methods

Statistical methods were used to check the detection of the location of faults. This was done by using the Box-Whisker method [50] (i.e. a box plot) which quantifies changes in the measurement of neighbour sensor nodes. With this method, the box represents the middle of the data while the median is the line around it at a range known as the inter quartile range. The box extends to the maximum and minimum data levels unless there are outliers; these are defined as being outside 1.5 times the inter quartile range. Those points are then represented with dots, as shown in Figure 6-17.

	Data set	Loss	Algorithm	Box- Whisker Method
Number of detected deviated data	**Intel Lab**	**38%**	101923 **confirm 64857**	160331
Number of none deviated data			649283	553809
Number of detected deviated data		**65%**	108133 **confirm 83891**	86246
Number of none deviated data			325639	323284

Table 6-4. Detection Comparisons between the Proposed Algorithm and the *Box-Whisker method*

Table 6-4 shows the number of outlier detections using the proposed algorithm and the Box-Whisker method for the Intel experiment data sets with 38% and 65% packet losses. The table shows that the proposed algorithm detects 101923 and 108133 deviations in the 38% and 65% loss data sets respectively. The 'Decision Confidence Control' module confirmed around 64% of these detected deviations to be deviations (in the data set for 38% loss). The remaining 36% were considered to be due to normal changes in the phenomenon and/or environmental effects. This is due, first, to the detection of similar levels of change between more than one neighbour and to a reduction in the proposed algorithm's confidence as a result of neighbour node packet losses, as discussed in Chapter 4. On the other hand, the Box-Whisker method did not detect these similarities or detect them as outliers.

However, with a set containing 65% packet loss, the algorithm detects 96% of the outliers detected by the Box-Whisker method. These experiments show that deviations detected by the proposed algorithm lie within the same outlier regions as those detected by the Box-Whisker method. In addition, the table shows that detection using the Box-Whisker method decreased by 46% when losses increased from 38% to 65%; at the same time, the detection increased by 6% and its confirmed detection increased by 30%.

6.4.3 Validating the Algorithm's Detection Using the Bayesian Based Algorithm

The Bayesian based algorithm was used to check the position of faults detected by the proposed algorithm. This is due to its practical method using the Motes self-healing hybrid sensor network architecture middleware, as in [23], [80].

Figure 6-18 illustrates the detection of the Bayesian based fault-recognition algorithm with a node failure probably threshold of 0.5[11] and the VMBA algorithm with a 30% detection threshold. Both algorithms are tested with the Intel Lab experiment data set [57] and operate at Node 1. The figure shows that the detected deviation data of the two algorithms were almost the same but the proposed algorithm detected more deviated data. This is due to the functionality of Bayesian fault-recognition in evaluating only its own readings while the proposed algorithm evaluates all readings in the neighbourhood.

Table 6-5 shows detection comparisons between the proposed algorithm and the Bayesian fault-recognition algorithm with several data sets. The table shows that the detection confirmation (i.e. the confirmation by the algorithm's 'Decision Confidence Control' module of the deviated data) of the proposed algorithm increased from 63% with 38% packet losses to 77% with 65% packet losses in the Intel lab experiment data set. In the garden experiment data set, however, this confirmation of detection increased from 93% with 0% packet losses to 95% with 14% packet losses. This means that the

[11] The calculated optimal threshold for the Bayesian based algorithm [20].

confirmation of detection increases as losses increase because losses reduce randomly the number of changes in similar measurements in any time interval.

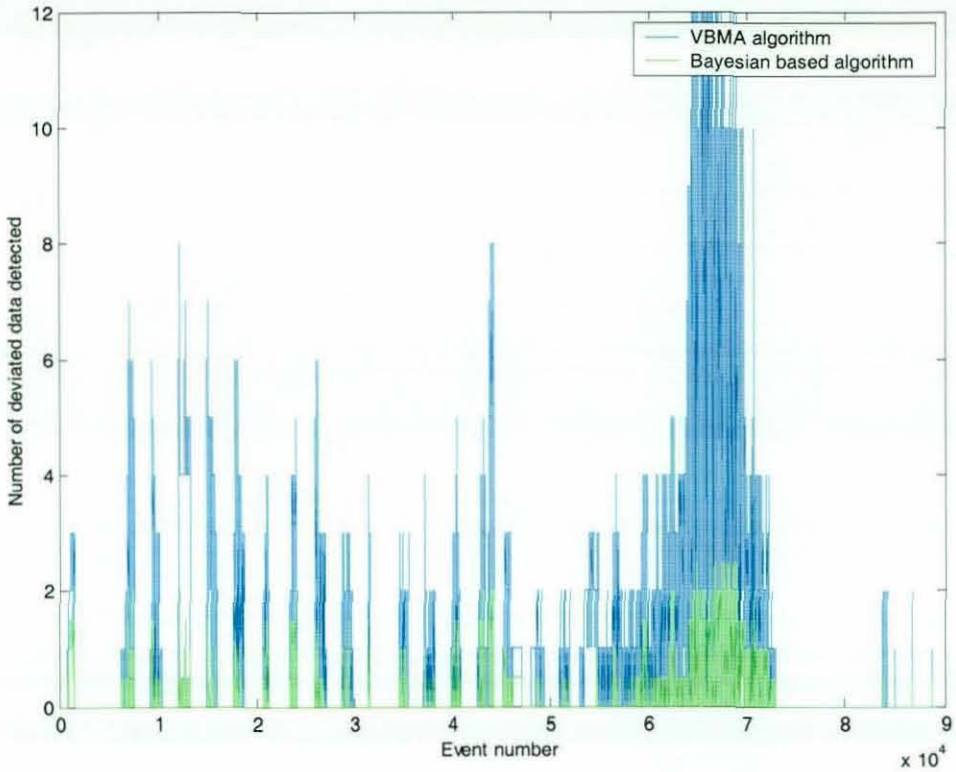

Figure 6-18. Fault Detection of the Proposed and Bayesian based Algorithms with 38% losses

Moreover, the table shows that the Bayesian based fault-recognition algorithm [23] detection increased three times when losses increased from 38% to 65% in the Intel lab experiment. This detection reduced in the garden experiment set by 12% when losses increased from 0% to 14%. This is because, with the garden data set, the changes were temporary and low, as a result of placing the sensors at different altitudes; (the analysis of the Bayesian algorithm depends on detecting high deviations).

In all data sets, the proposed algorithm's detection of node 1 deviation was, on average, 20% higher than the Bayesian detection of fault-recognition for all data sets. This is due to the ability of the proposed algorithm to detect more than one deviation at the same time interval.

	Data set	Loss	Algorithm	Bayesian fault-recognition algorithm
Number of detected deviated data	**Intel Lab**	**38%**	101923 **confirm 64857**	1623
Number of none deviated data			649283	712517
Number of detected deviated data		**65%**	108133 **confirm 83891**	5283
Number of none deviated data			325639	404247
Number of detected deviated data	**Garden**	**0%**	2979 **confirm 2797**	290
Number of none deviated data			57516	60023
Number of detected deviated data		**14%**	2882 **confirm 2752**	254
Number of none deviated data			56808	59306

Table 6-5. Detection Comparisons between the Proposed Algorithm and the Bayesian Fault-recognition Algorithm for Different Data Sets

6.4.4 The Algorithm's Detection of Permanently Deviated Nodes

Figure 6-19 illustrates the effect of increased percentages of faulty deviated nodes on the proposed algorithm's detection per event for 100 runs with 1,000 nodes each with a 50 metre transceiver range. These nodes are randomly deployed over a 1000X1000 square metres with 0.1% packet losses, 0.1% random deviated measurements, and 0.1% dead nodes. The figure shows that, as the percentage of deviated faulty nodes increases, the proposed algorithm's detection of faulty nodes decreases exponentially and reaches around 20% when faulty deviated nodes reach a level of 80%. The algorithm's positive false detection increases linearly as the number of deviated faulty nodes increases, reaching around 80% when 80% of the network's nodes are faulty.

Unfortunately, this type of positive false detection is very difficult to control due to its dependency on the number of deviated nodes in the neighbourhood

and the number of neighbours for each monitoring node. The proposed algorithm depends on its confidence control and passive warning tests to reduce this positive detection.

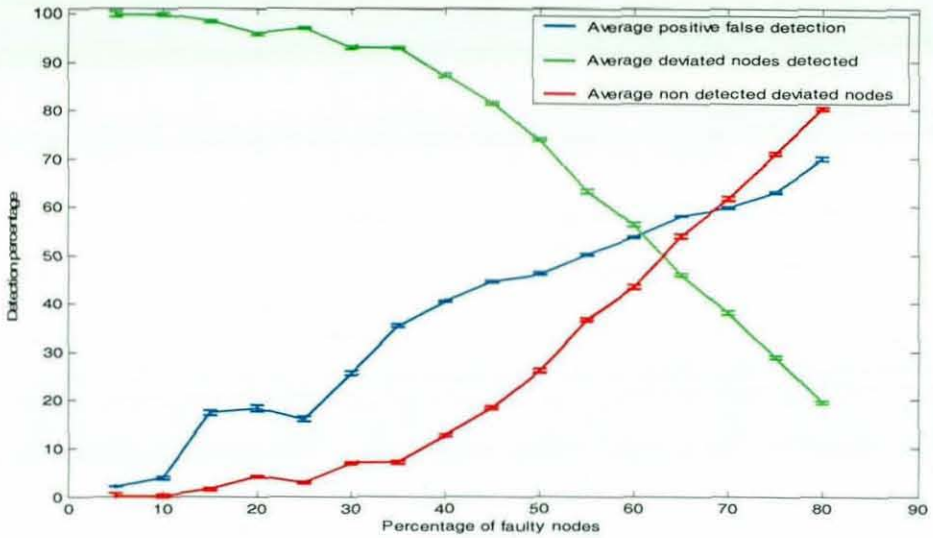

Figure 6-19. The Algorithm Detection versus the Percentage of Faulty Nodes

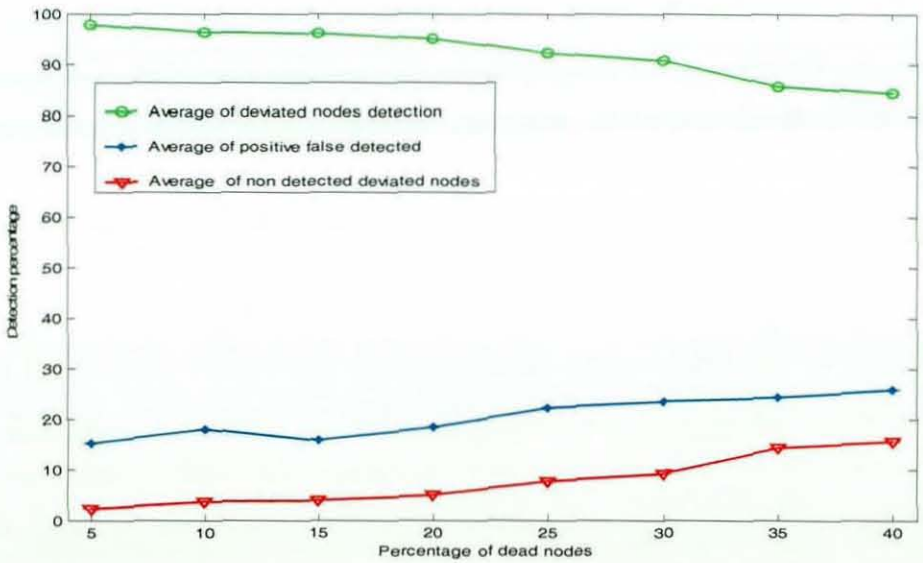

Figure 6-20. Detection of Deviated Nodes versus the Percentage of Dead Nodes

6.4.5 Detection of Dead Nodes

Figure 6-20 illustrates the effect of an increased percentage of dead nodes on the proposed algorithm's detection per event for 100 runs with 1000 nodes with a 50 metre transceiver range randomly deployed over a 1000X1000 square metre area with 0.1% packet losses, 20% deviated nodes and 0.1% random deviated measurements. The figure shows that, as the number of

dead nodes increases, the detection of deviated nodes slowly decreases with a speed depending on the position of the dead node.

6.4.6 Comparisons between Methods Used for Evaluation of the Proposed Algorithm's Detection

In order to test the consistency of different methods of detection for the proposed algorithm, comparisons were made between them for a deviation detection of 10 nodes[12], as shown in Figure 6-21. The metric used for evaluating the detection of the three methods was the ratio of detected faults compared to the total number of faults. This metric illustrates the consistency of the three methods used to detect deviations.

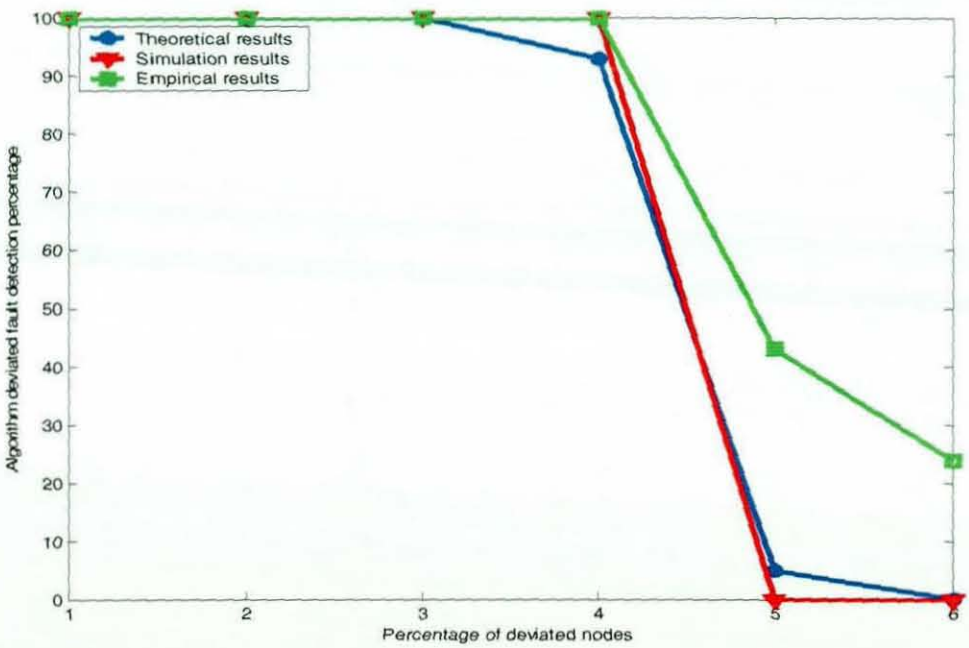

Figure 6-21. Comparison between the Detection of the Three Evaluation Methods

Figure 6-21 shows that the detection percentage of the simulation, the empirical and the analytical tests were almost the same. There were two differences between them: the first was that, after 40% of nodes were deviated, the analytical model detection was reduced to 90%, while the empirical and simulation models remained at 100%. The second difference was that, after 50% of nodes had deviations, the empirical and analytical

[12] Please note that the number of nodes selected depends on the testbed size

models indicated a percentage of detection of faulty nodes while the simulation model calculated 0% of faulty node detections. This shows the high consistency of the results using the three evaluation methods.

6.5 Proposed Algorithm for the Performance Evaluation of *WSN* Event-Driven Applications

One very important class of *WSN* applications is event-driven; this is where the application functions vary due to the size and location of the event that is detected. This class of application is a new challenge to the proposed algorithm, especially in tracking the health of nodes at an event boundary.

The implementation of the event-driven algorithm 'method 1' was compared with the Bayesian algorithm proposed in [23] in simulation experiments. This is because the proposed algorithm is similar to the Bayesian algorithm proposed in [23] and the Bayesian algorithm used for the health detection of *WSNs* [80]. The metrics used to analyse the results were the percentages of positive and negative false detections. The first is the ratio of healthy nodes detected by the algorithm as faults, compared to the total number of detections. The second is the ratio of undetected faults as opposed to the total number of faults in the network. The two metrics show the algorithm errors for different scenarios.

The experiments used 1000 random nodes deployed over a 1000X1000 square metre area with a 50 metre transceiver range. These experiments showed a constant positive detection rate of 10% when faulty nodes in the neighborhood reached 10%. The Bayesian-based algorithm showed a good level of detection up to 3% but then negative fault detection started to increase as the number of faulty nodes increased. This is shown in Figures 6-22, 6-23 and 6-24. From these experiments, the conclusion could be drawn that the proposed algorithm has better performance than the Bayesian approach in terms of fault detection as, with a continuous positive false detection of 10%, this can be reduced by the use of a confidence control but it increases the level of negative detection.

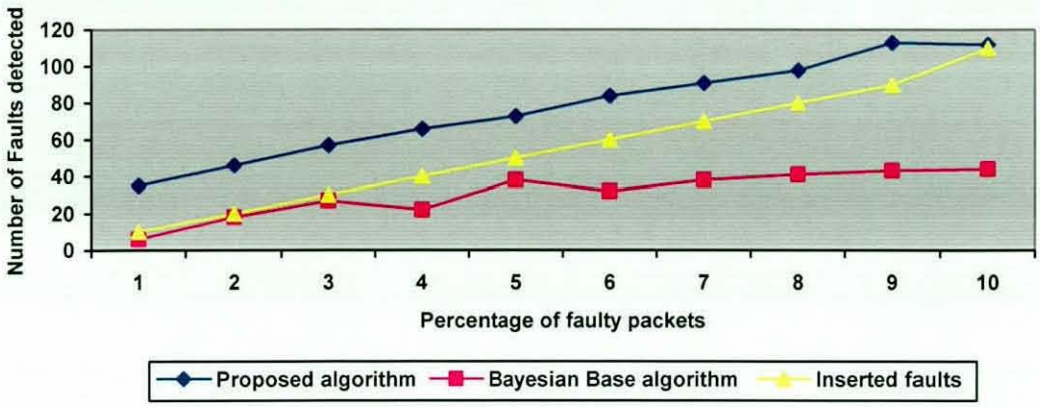

Figure 6-22. Proposed Algorithm's and Bayesian Algorithm's Fault Detection.

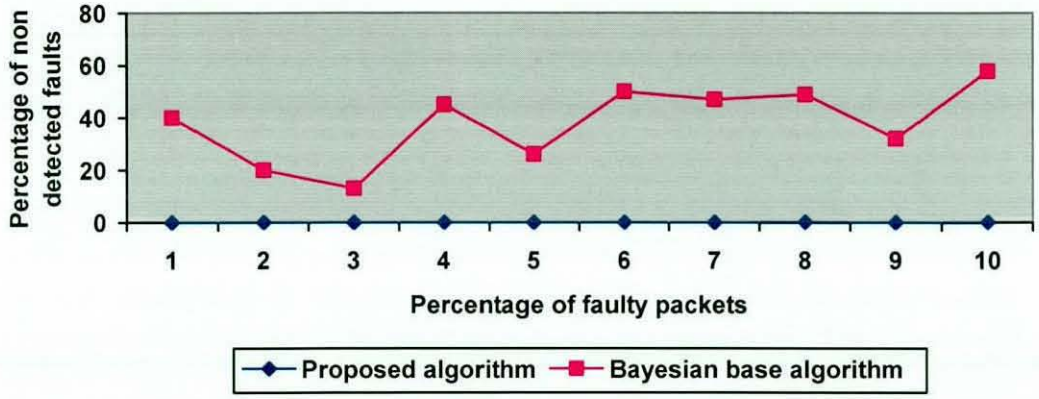

Figure 6-23. Non-Detection Percentage of the Proposed Algorithm and Bayesian Based Algorithm

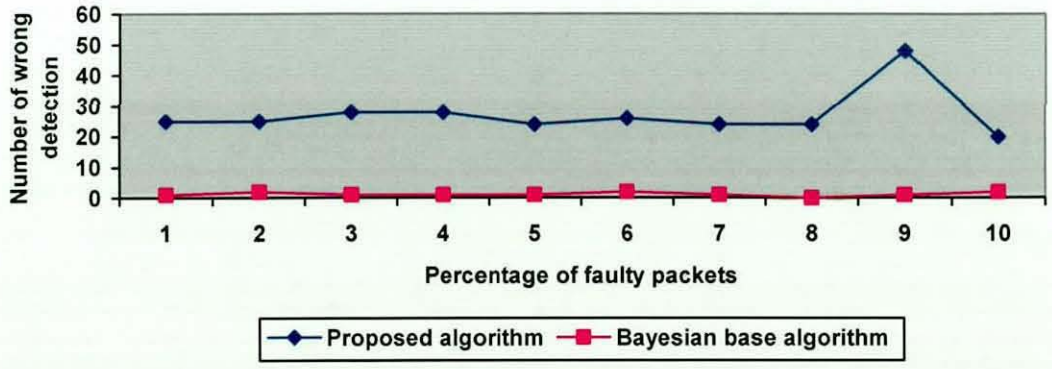

Figure 6-24. Wrong Detection Percentage of the Proposed Algorithm and Bayesian Based Algorithm

Also, the *VMBA* algorithm shares a number of similarities with the collaboration fault detection algorithm proposed in [56]. The collaboration algorithm is one that detects deviated and dead nodes in the network by sending consult packets to neighbours when faults are detected. This is done by tracking the difference between monitoring node measurements and each

of the neighbour measurements, and then comparing the difference with a threshold. If the difference exceeds that threshold, then the monitoring node sets on a warning timer and sends a packet (i.e. a consult packet) to the neighbours of the suspected node requesting them to send their analysis of this node. (This is carried out through a route constructed around the suspected node.) If the neighbour nodes' analysis is greater than the monitoring node's analysis (i.e. the difference in measurement is higher than the threshold) then the monitoring node will release a warning packet to indicate the detection of a suspected node. Otherwise, it will reset the warning timer and cancel the warning.

When the same experimental scenarios described above were repeated, the results of the experiments showed that the collaboration algorithm functioned with a large amount of traffic which consumed more power for transmission, receiving, memory storage and processing; the amount of this consumed power depends on the routing protocol used. Although the amount of positive deviations detected by the proposed algorithm was more than with the collaboration algorithm, it works better at higher fault percentages and with lower resource usage, as shown in Figures 6-25 to 6-30.

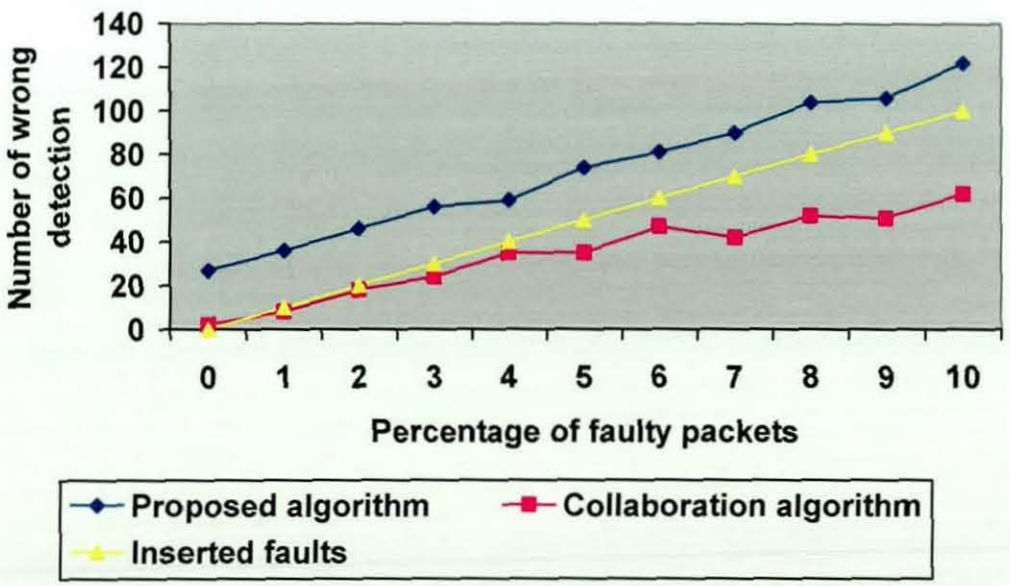

Figure 6-25. Proposed Algorithm's and Collaboration Algorithm's Fault Detection

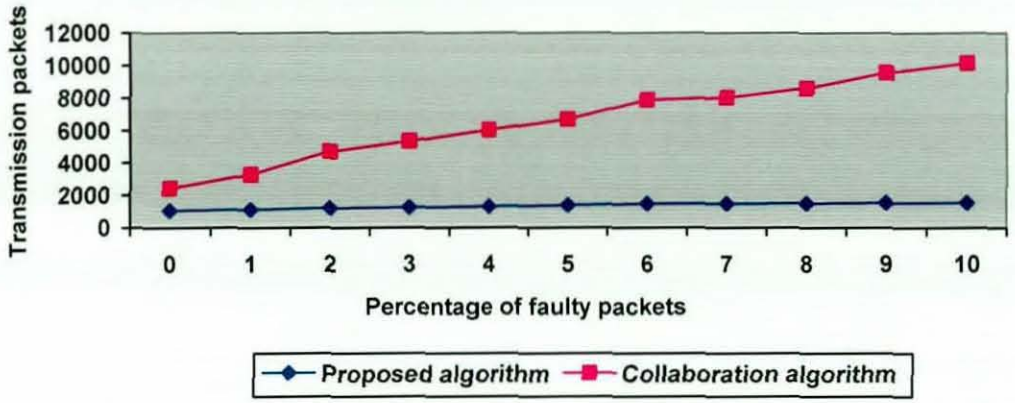

Figure 6-26. Transmission Packets from the Proposed Algorithm and the Collaboration Based Algorithm

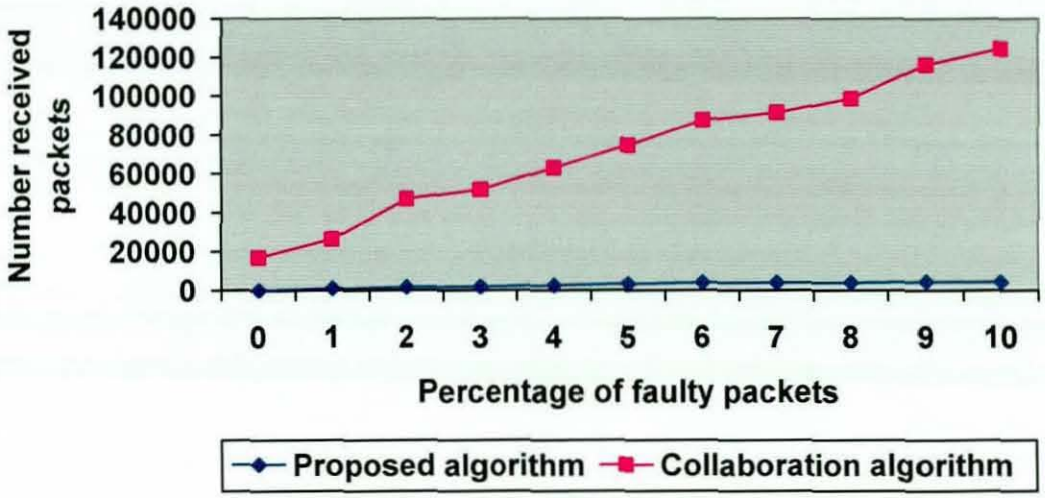

Figure 6-27. Received Packets in the Proposed Algorithm and the Collaboration Based Algorithm

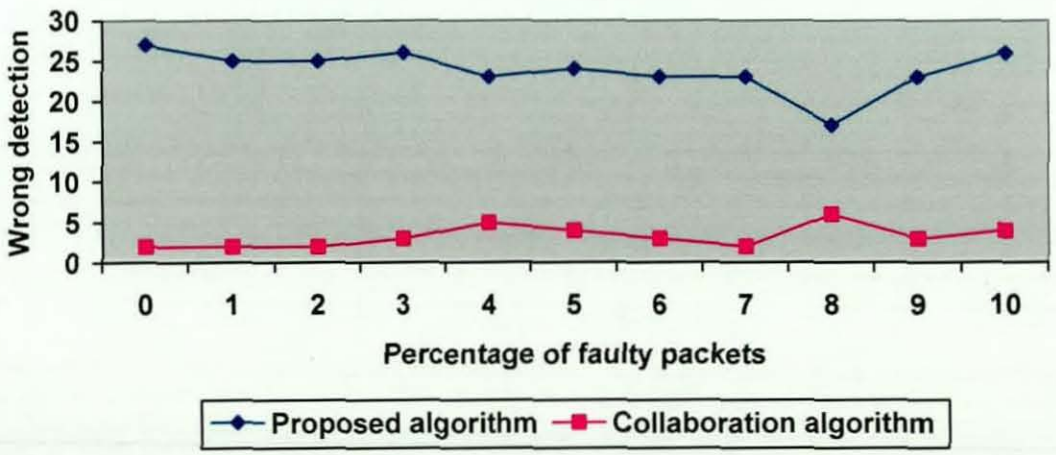

Figure 6-28. Wrong Detection Percentage of the Proposed Algorithm and the collaboration Based Algorithm

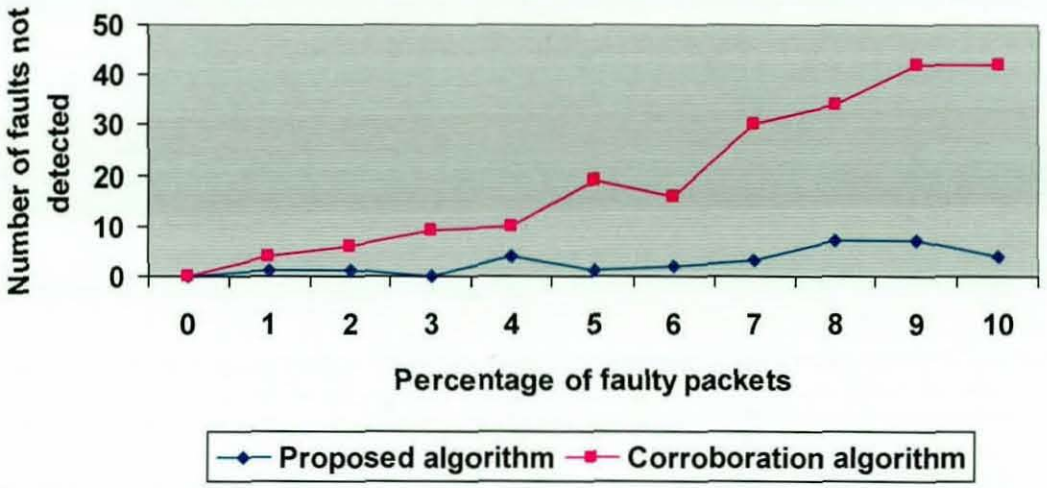

Figure 6-29. Non-Detection Percentage of the Proposed Algorithm and the Collaboration Based Algorithm

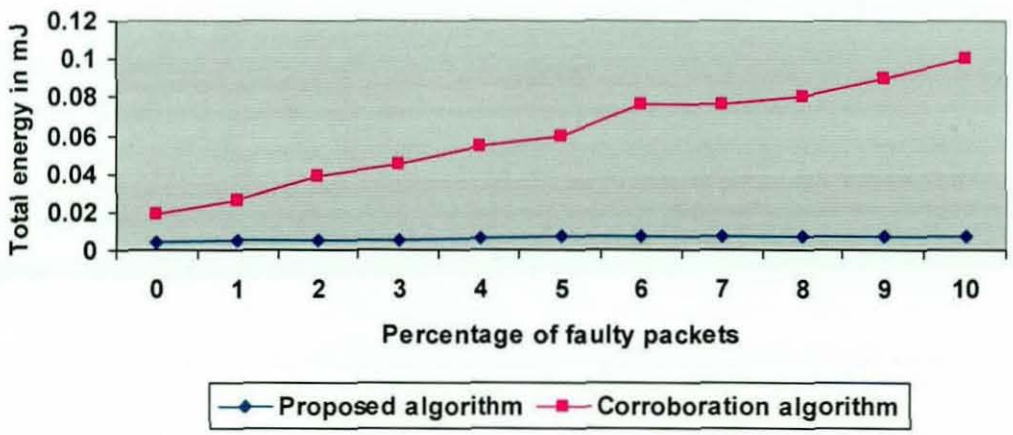

Figure 6-30. Total Energy Spent in the Proposed Algorithm and the Collaboration Based Algorithm

6.6 VMBA Algorithm Limitations

The VMBA algorithm uses a passive voting technique that assumes that any reading which is different from a majority is a change, either due to a fault in the sensor node, sensor battery depletion, or a network coverage problem. As a result of this simple technique, the algorithm's functionality depends on three main factors: the number of neighbours that participate at an event, the number of deviated measurements at a monitoring time interval, and the degree of deviation of the measurement.

6.6.1 Number of Neighbours

The number of neighbour nodes depends on many factors in *WSNs*; these include: the coverage, connectivity and redundancy required by the application; receiving and sensing ranges; and node transmission power. The selection of neighbour nodes is controlled by protocols in order to limit communication costs and to reduce computational complexity. The neighbour selection of these protocols is based on the history of forward packets, the connectivity, the quality of links, power consumption, and the close geographic proximity of nodes.

The number of neighbour nodes is very important in the proposed algorithm where this adds more confidence to the algorithm detection performance, as is discussed in the next section. However, this increases energy consumption and generates more traffic in the network.

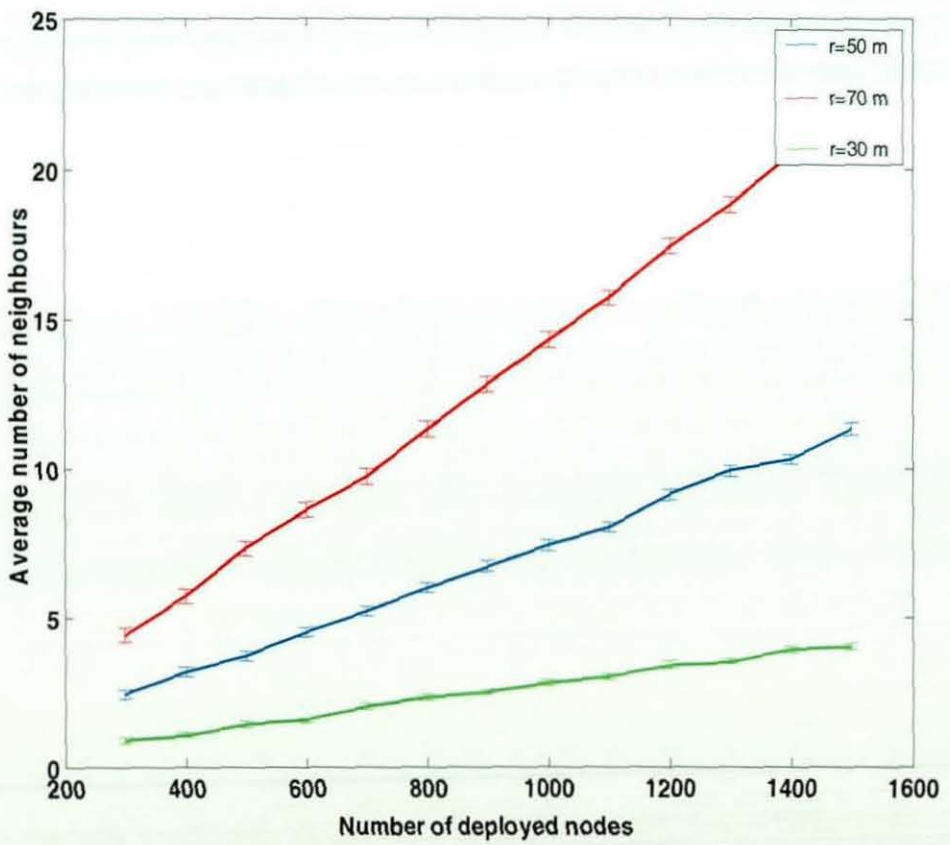

Figure 6-31. Number of Neighbours versus Deployment Density with Three Transmission Ranges

Figure 6-31 shows the relation between the number of neighbours and the number of randomly deployed nodes detected. This was generated from100 runs of a simulation of deployed nodes in a 1000X1000 square metre area with different node transceiver ranges. The figure shows that that number of neighbours depends on the node density in the area and the communication range. This average number of neighbours linearly increases as the density of deployment increases; but at the same time, the nodes' average power consumption percentage increases almost linearly after the second neighbour, as shown in Figure 6-32.

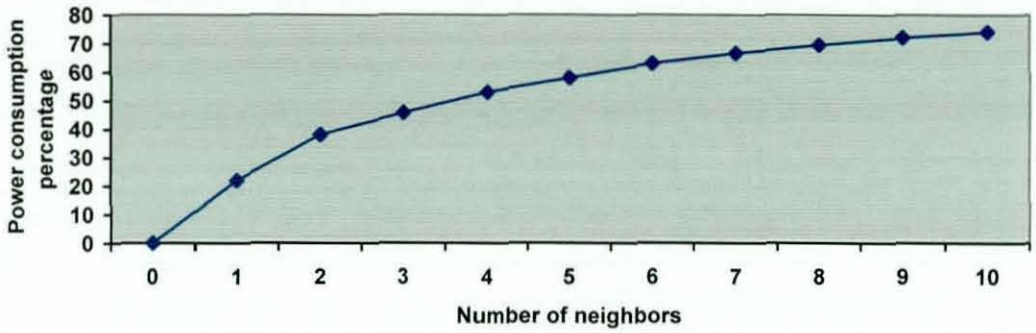

Figure 6-32. Percentage of Total Expected Node Life Time Reduction with Reference to Number of Neighbours

6.6.2 Number of Nodes and the False Detection of Faulty Nodes

The minimum number of measurements that the algorithm can work with is two. With this number there is an indication of a deviated reading but the algorithm cannot isolate the faulty one; when a large-scale Wireless Sensor Network reaches this situation, the probability of network disconnection will be very high while the probability of covering the targeted phenomenon will be very low.

Figure 6-33 illustrates the relation between the false detection rate and the number of deployed nodes for 100 runs with a 50 metre transceiver range deployed over a 1000X1000 square metre area using different densities. The figure shows that as the density of nodes increases, the probability of false detection in the algorithm reduces. This is due to the increase in the number of neighbours which increases the detection confidence. The probability of

false detection reaches almost 0 when there are more than 8 neighbour nodes.

6.6.3 Loss and Error in Detecting Faulty Nodes

The number of received neighbour packets may vary during network operation due to losses. This is shown clearly in the experiments when 1000 nodes were randomly deployed in a 1000X1000 square metre area, each with a 50 metre transceiver range. When the losses in the network increased from 30% to 90%, the percentage of occasions when a packet was not received from neighbour nodes increased from 0% to 30%, as shown in Figure 6-34. In addition, the figure shows that there is a high variation in the confidence intervals of the packet losses. This confidence interval increases as packet losses increase and these losses influence the confidence level of the calculated neighbourhood median, as shown in Figure 6-35.

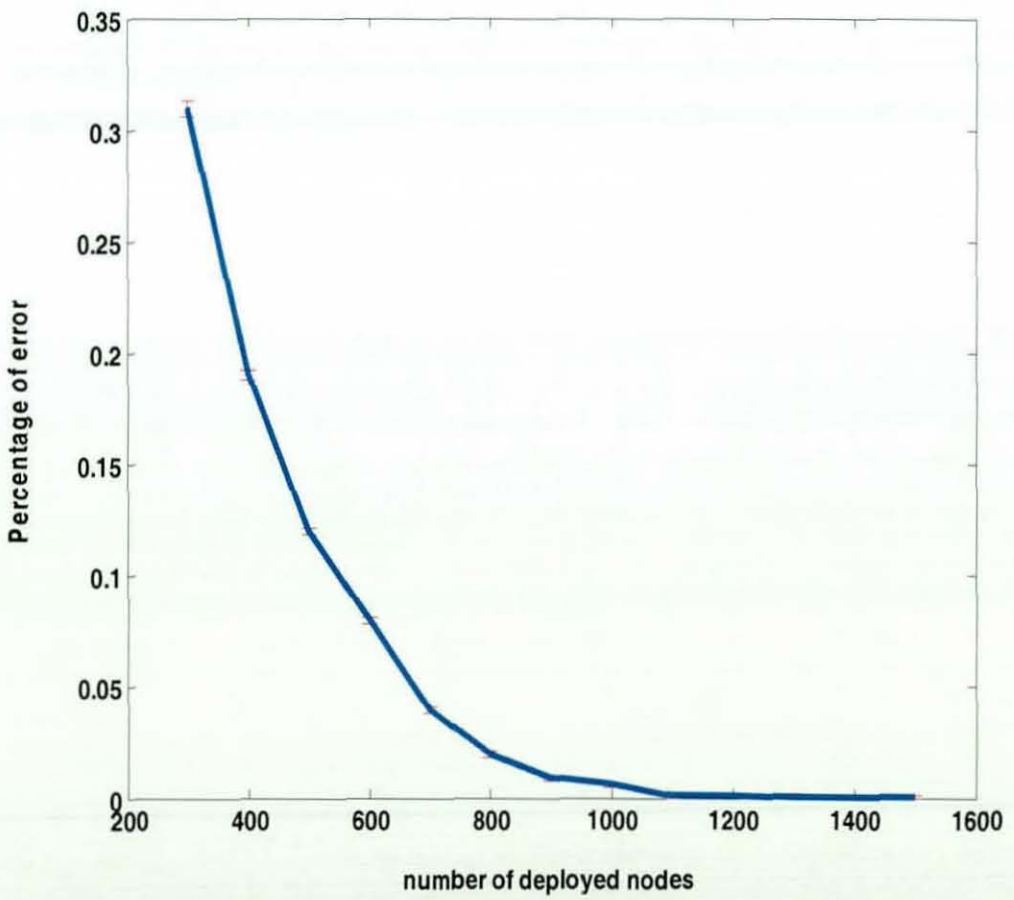

Figure 6-33. Error Reading Deviation Detection Due to Less Than Two Neighbours

The variation in median confidence increases the probability of error detection. This is because, when the number of permanent deviated nodes in the neighbourhood increases, the probability of their impact on the neighbourhood collaboration function also increases. Furthermore, the existing high level of random packet losses between nodes tends to make them a majority at some time intervals, as shown in Figure 6-36. This figure also shows the calculated median for the simulation experiments conducted with a simulated data set taken from the UC Berkley Botanic Garden set [58], as discussed in Chapter 7. The figure shows clearly that the losses affect the calculated median, especially when the number of deviated nodes becomes the majority. This limitation has been solved by adding a validation test to module 2 in the proposed algorithm. This test compares the two consecutive median values to the expected change in the phenomenon, as was explained in Chapter 4.

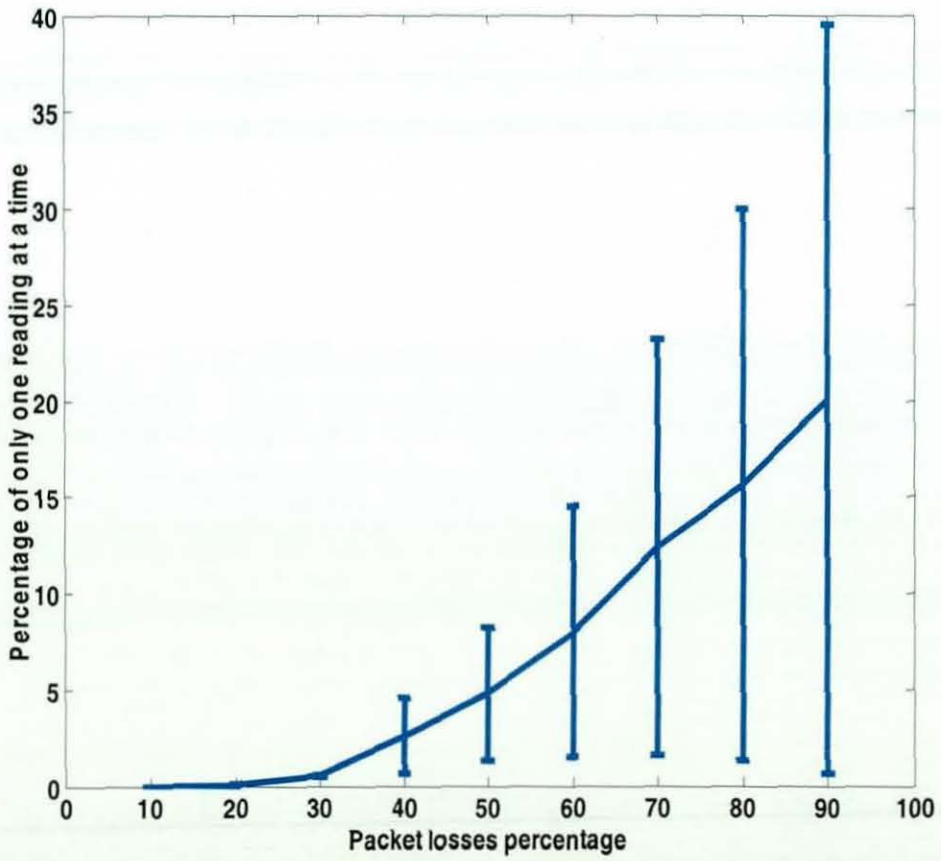

Figure 6-34. Percentage of losing all neighbour packets verses network packet loss

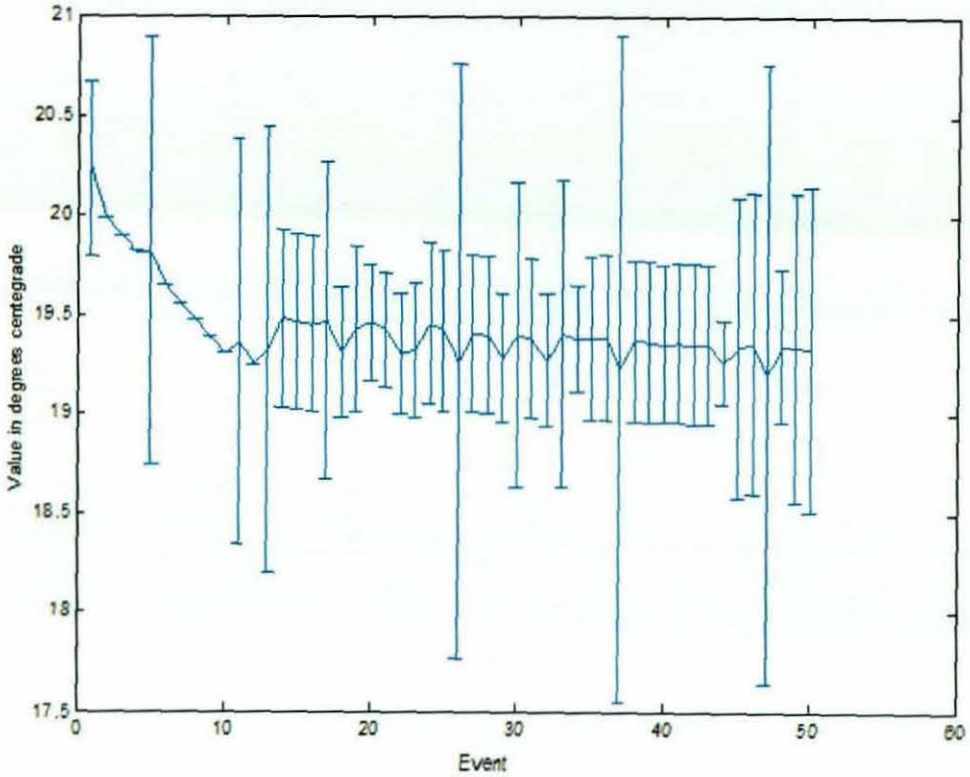

Figure 6-35. The Change of Median Confidence Interval with Losses

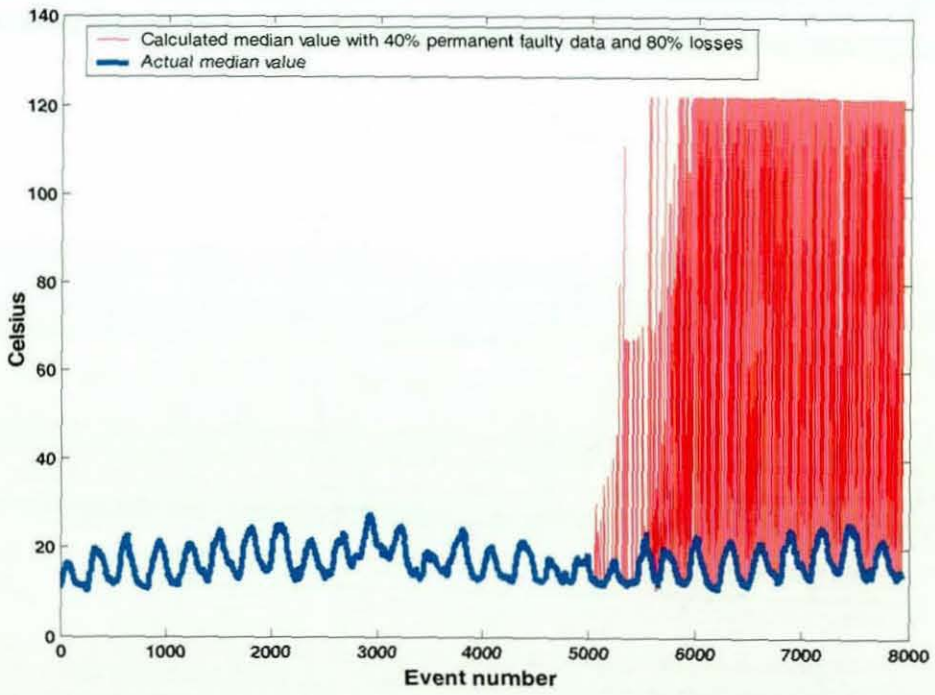

Figure 6-36. Median Calculation under Losses Medium

6.7 Summary

This chapter justifies the functionality of the proposed algorithm in terms of the prediction of the phenomenon's measurements, the algorithm's deviation detection, and its resource usage. It shows that the algorithm's prediction values for the phenomenon are within a maximum error of 4% with 65% losses medium when compared with algorithms that use more complex analysis and models. Moreover, it shows that the power consumption, as a result of the algorithm's usage, reaches 0.8% of *CPU* energy consumption. At worst case, it reduces the *'Surge'* application network's lifetime with an average of 16 neighbour nodes by 0.01%.

In addition, the chapter discusses some drawbacks that were detected in the algorithm functionality using simulation and empirical experiments. These limitations are due to the voting low complex technique that was used and the fact that its analysis depends on the number of healthy readings, the number of neighbour readings in the median calculation, and packet losses.

Chapter 7 VMBA Simulation Experiments

7.1 Introduction

This chapter discusses the Voting Median Base Algorithm (*VMBA*) which makes approximate measurements of Wireless Sensor Network performance and its' functionality at both the network and the node levels. At the network level, the algorithm was tested by tracking its percentage of both positive and negative false detections. At the node level, it was evaluated by testing its detection of permanent and temporarily deviated data, the effect of this on the accuracy of the data collected in the neighbourhood (i.e. the percentage of closeness between the neighbour readings), and the effect on the network's performance before the algorithm detects the deviation.

The evaluations were conducted with two different types of data set (i.e. simulated and real world data) with different measurement characteristics and different losses. These data sets were used to check the impact of different scenarious on the functionality of both the network and the algorithm in order to ensure that the experiments tested the algorithm and not the data.

The chapter starts by listing the assumptions used in the simulation experiments that were conducted. It then discusses the outcomes of these simulations and analyses the algorithm's functionality using the simulated data sets at both network and node level. This is followed by a discussion of the simulation results using real data sets. Finally, the chapter discusses two methods that are proposed for reducing the proposed algorithm's power consumption.

7.2 Simulation Assumptions

Several assumptions were made in these simulation experiments. They are as follows:

- All nodes have the same characteristics (i.e. they are homogenous), so that they have the same capability of running and executing the

algorithm's code. (If they were not, then the algorithm would work at the cluster head node).

- Sensor nodes are static and each of them has a unique identification; i.e. *ID*. (This is due to the proposed algorithm's need to track and analyse neighbour nodes individually.) The only mobility here considered is the high topology dynamic due to losses (node mobility is one of the future extensions to this study).
- The network is synchronised and the data table is constructed by the application. This is because the algorithm follows the application's flow process and uses the application's parameters in its functionality to reduce code complexity and processing time.
- Nodes in the neighbourhood have a constant communication range; they can hear each other within it and their packets that broadcast to immediate neighbours are subject to collision (i.e. there is no acknowledgment or re-transmission). This is done in order to increase the effect of losses on the accuracy of the collected data and the network's functionality.

7.3 Results from Simulated Data

Simulated data were created as discussed in Chapter 2 to check the proposed algorithm's deviation detectability at both network and node levels. This was done by using two types of simulated data. The first type was created to assess the detection capability of the algorithm at the network level, together with the effect of random deviated data, packet losses, dead nodes and faulty deviated nodes on the algorithm's functionality. The second simulated data set used a sample of 11 real world temperature measurement patterns taken over 48 hours in UC Berkley Botanic Garden [58] to model the correlation between different nodes. Then, different scenarios of random data deviations, permanent node deviations, packet losses and dead nodes were superimposed over the original data in the data set to modify it according to predefined percentages, nodes and events.

The experiments for each set of random losses were repeated numerous times to test the impact of distribution of random losses on the algorithm's functionality.

7.3.1 Network Level Simulation Results Using Simulated Data Sets

The experiments for this data set employed MATLAB as a tool in simulating scenarios using 1000 sensor nodes randomly distributed over a 1000X1000 square metre area, each with a 50 metre transceiver range. During the simulation, the position of each sensor was fixed and each was run with the same *VMBA* algorithm to detect sensor deviations. Three metrics were chosen in order to analyse the results of these experiments. The first metric was the average percentage of algorithm detection; this measures the average ratio of the number of deviated faulty sensors diagnosed as faulty as opposed to the total number of faulty deviations in the network. This metric computes the capability of the *VMBA's* fault detection. The second metric chosen was the algorithm's average positive false detection, which measures the average ratio of the number of non-deviated faults diagnosed by the algorithm as faults, compared with the total number of deviated faults that are diagnosed. This metric computes the positive errors detected by the algorithm. The last metric chosen to evaluate the results was the algorithm's average of negative false results. This is the average ratio of deviated faults that were not diagnosed as such, as opposed to the total number of faults. This metric defines percentage of deviated faults in the network that were not detected.

Each experiment is the result of 100 events of a random percentage of data deviations and packet losses.

7.3.1.1 Detection of Deviated Faulty Nodes

There are many factors that affect the proposed algorithm's detection of permanently deviated nodes. One of these factors is the deviated data received per event. Figure 7-1 illustrates the effect of increasing the percentage of deviated data per event on the proposed algorithm's detection

with 20% permanently deviated faulty nodes, a 0.1% packet loss, and with 0.1% dead nodes. The figure shows that when deviated data were fewer than 10%, the algorithm's detection per event was steady at around 97%. When this deviated percentage increased above 10%, the algorithm's detection per event went down linearly until it reached 60% with 40% of deviated data per event. This occurred along with a high increase in the algorithm's positive detection. This reached 80% of the algorithm's total detection per event at a 40% level of deviated data (i.e. 60% of deviated data per event). This positive detection was reduced by increasing the algorithm's detection confidence using a monitoring window and data validation tests, as explained in Chapter 4.

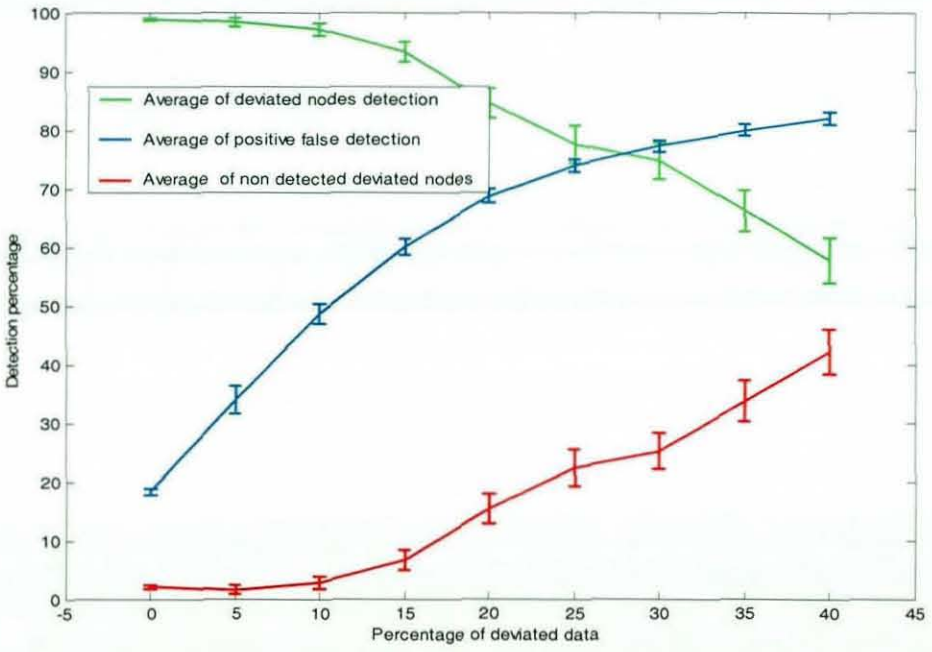

Figure 7-1. The Algorithm Detection of Faulty Deviated Nodes with Different Deviated Data Percentages, 0.1% Packet Loss and 0.1% Dead Nodes

Figure 7-2 illustrates the proposed algorithm's positive detection when a monitoring window was set with 10 samples and with a 6 sample window threshold. The figure shows a reduction of around 40% of positive warning messages between using and not using the monitoring window. However, this reduction came along with a 5% reduction in the algorithm detection of faulty deviated nodes.

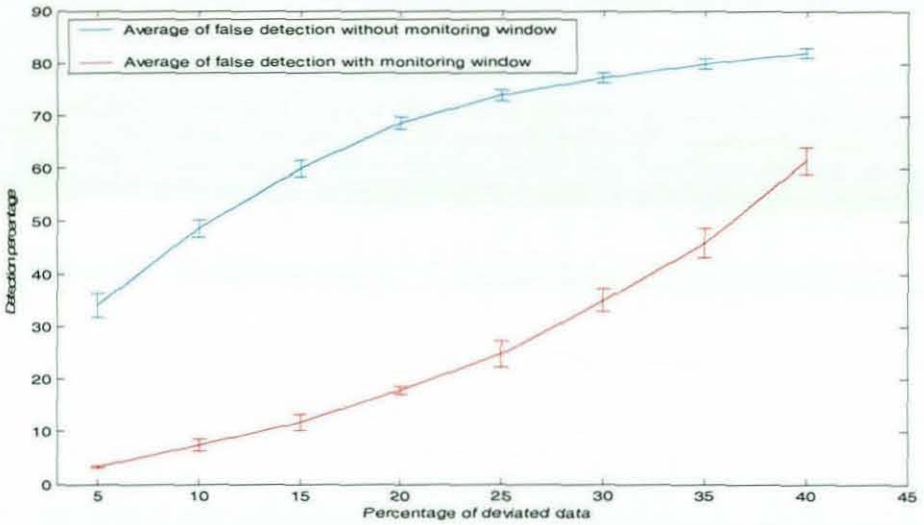

Figure 7-2. The Algorithm Detection of Faulty Deviated Nodes with Different Deviated Data Percentages, 0.1% Packet Loss and 0.1% Dead Nodes

Another factor that affects the proposed algorithm's detection is packet losses. Figure 7-3 illustrates the effect of increasing the percentage of packet losses on the proposed algorithm's detection with 20% of faulty deviated nodes, a 0.1% random deviated data, and with 0.1% dead nodes. The figure shows that, as losses increased, the algorithm's detection decreased linearly and reached 40% at a 60% loss. This occurred along with a gradual increase in the algorithm's positive detection that reached 20% with 60% losses. If a monitoring window of a sample size of 10 and a 6 sample threshold was used, the positive detection was reduced to almost 0%, as shown in Figure 7-4.

7.3.1.2 *Detection of* Dead Nodes

The algorithm's detection of dead nodes is mainly affected by the percentage of packet losses, as discussed in Chapter 4. Figure 7-5 illustrates the effect of losses on the detection of dead nodes per event with 20% dead nodes, 0.1% faulty deviated nodes, and 0.1% random deviated data. The figure shows that, as the losses increase, the percentage of the algorithm's detection of dead nodes remains stable but its positive detection increases exponentially and reaches 65% of the total algorithm's detection per event with a 50% random loss (i.e. with 70% of data loss). This positive detection is reduced to 0%

when a monitoring window of a sample size of 10 and 100% window threshold is used.

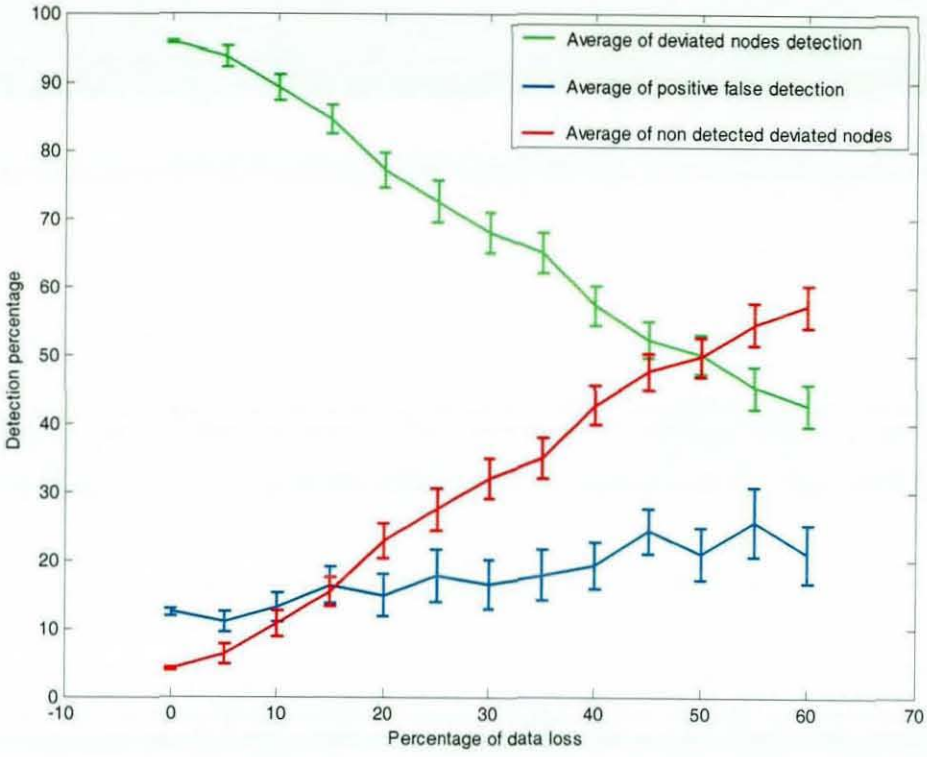

Figure 7-3. The Algorithm Detection of Faulty Deviated Nodes with different Data Loss Percentages and 0.1% Dead Nodes

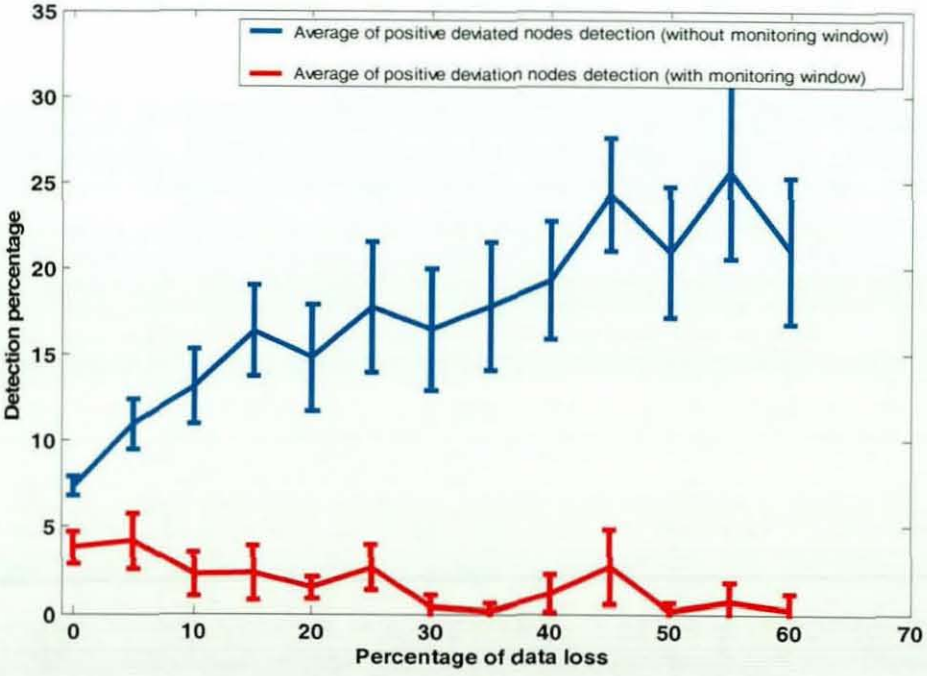

Figure 7-4. The Algorithm Detection of Faulty Deviated Nodes with Different Data Loss Percentages and 0.1% Dead Nodes

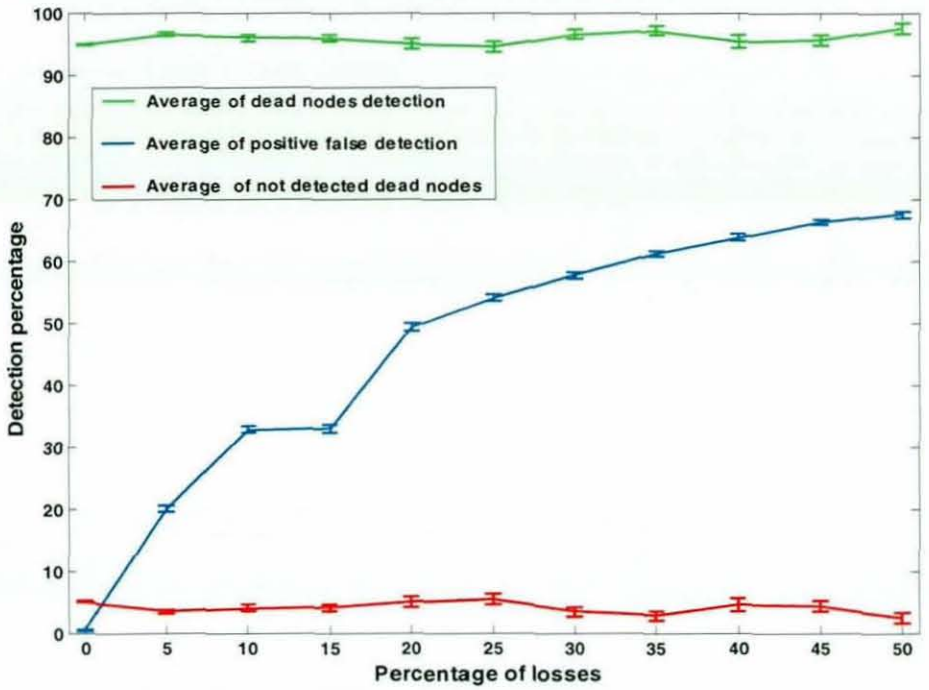

Figure 7-5. The Algorithm Detection of Dead Nodes with Different Data Loss Percentages and 0.1% Deviated Nodes

7.3.2 Node Level Simulation Results Using Simulated Data Set

These simulations were set with known percentages of permanently deviated faults to test the performance of the proposed algorithm. To test these data sets, the algorithm was implemented in *MATLAB* on node 1 such that it could listen to the communication of the other 10 nodes. The aim of these experiments were to test the impact of deviated sensor readings, neighbour packet losses, monitoring window size and threshold, and the removal of the detected nodes confirmed as faulty on the accuracy of the neighbourhood's collected data, the network's performance, and the algorithm's detection at node level.

Three metrics were chosen to analyse the results of the experiments, as shown in Table 7-1. The first metric is the residual value of the deviated node, which is the difference between the neighbourhood median and the data at a time instance. This metric computes the diversity of individual readings from nodes as compared with readings from other nodes in the neighbourhood. In

addition, it shows the behaviour of the fault. This metric can be compared with weighted residual metrics: i.e. the difference between a reading and the median calculated at a time instance multiplied by one minus the ratio of similar correlated readings compared to the total number of readings at the same time instance. This comparison shows the weight of each deviation on the neighbourhood's collected data.

Metric	Formula used
Weighted residual values of deviated neighbours	$\frac{\lvert Node\ measurements - Median \rvert}{Median} * \left(1 - \frac{Number\ of\ neighbourhood\ measurements\ correlate}{Total\ \ number\ \ of\ readings}\right) * 100$
Network performance	$\frac{Number\ of\ healthy\ readings}{Total\ Number\ of\ neighbours}$
Readings performance	$\frac{Number\ of\ healthy\ readings}{Total\ Number\ of\ readings}$

Table 7-1. Metrics Calculated During the Experiments

The second metric was the network performance, which is the ratio of healthy readings as opposed to the total number of nodes in the neighbourhood (i.e. without losses). This metric computes the effect of losses on the network's functionality. The last metric used in the evaluation of results was the reading performance (or reading reliability). This is the ratio of healthy readings compared with the total number of readings available at that event. This metric computes the reliability of the collected data in the neighbourhood and can be given in a continuous scale by using reading confidence, as used in [105], or by using formula (7.1).

$$\text{Reading performance} = \left(1 - \sum_{i=1}^{n} \frac{\lvert Mean\ of\ readings\ with\ deviated\ node - Mean\ of\ readings\ without\ it \rvert}{Mean\ of\ readings\ with\ deviated\ node}\right) * 100 \qquad (7.1)$$

Figure 7-6 shows the functionality of the network when the proposed algorithm is used without a classifier in a lossless medium and without removing the detected faulty node from the network. As can be seen from the figure, both the network and the reading performance have the same values as a result when no loss occurs. When a fault occurs, the reading

performance metric (i.e. reading accuracy) is reduced by 10% and remains at the same level until another node failure occurs. The neighbourhood's reading accuracy is then reduced until it reaches 57% after the fifth node's permanent deviation. This degradation was due to the effect of permanently faulty deviated nodes on the accuracy of the neighbourhood's collected data. This can be seen from the residual values that increased suddenly and very sharply at the occurrence of the first deviated faulty node. When another deviated faulty node occurred, the residual value was reduced due to the reduction impact of each deviation on neighbourhood reading accuracy. This change in the residual will affect any data-gathering technique, with the degree of its effect depending on the application's tolerance to residual value, its duration, the number of neighbour nodes, and the percentage of losses.

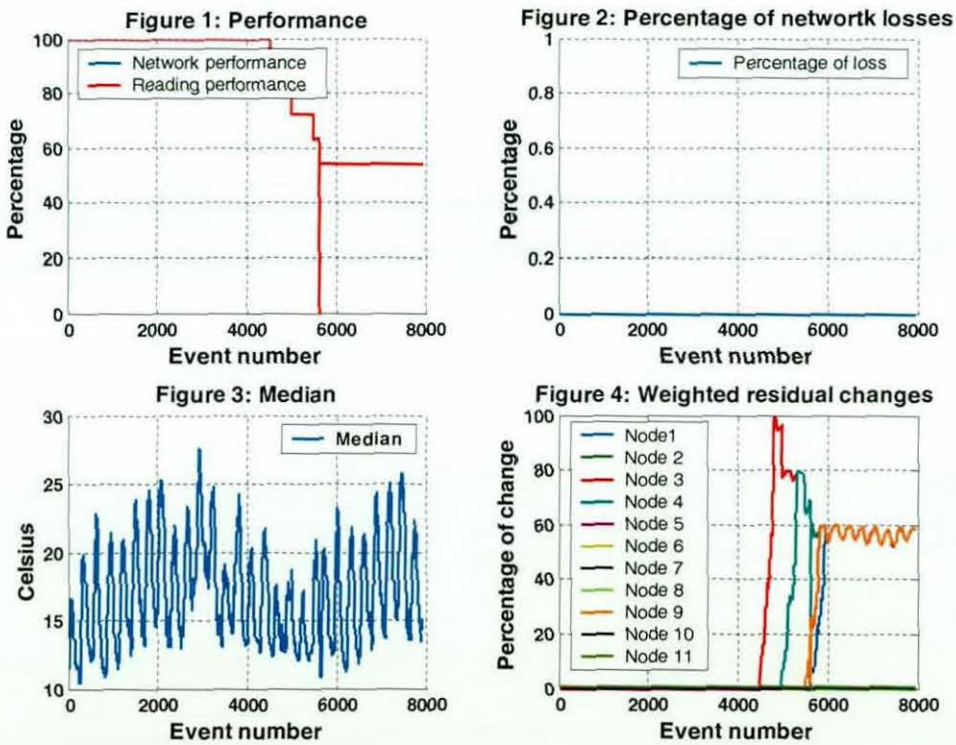

Figure 7-6. Simulation without loss and without isolating the deviated nodes.

Also, the figure shows a sudden degradation of reading accuracy to 0% at event 5800 due to the occurrence of more than one fault at the same time. This experiment showed that the degradation in reading accuracy depends on the value of deviation from the neighbourhood readings and the number of

deviated faulty nodes. These results are consistent with the detection described by Mehranbod in [64] which used a probabilistic approach.

Analysing the same data from the perspective of the proposed algorithm analysis (i.e. measurement weighted residual) shows that the tested data had 5 permanent faults (i.e. 2,3,4,8 and 9), two groups of coverage (first was 1,2,3,4,6,7, and the second was 5, 8,9,10), and a temporary deviated node (i.e.11), as shown in Figure 7-7. Also, the figure shows that, after the occurrence of more than one fault the weighted residual has reduces, and as the number of faults increases, the deviated node residual becomes less than the set threshold. Finally, a comparison of Figures 7-6.4 and 7-7 shows that the 5 permanent faults are detected in the algorithm at the same time as their residual changes grow.

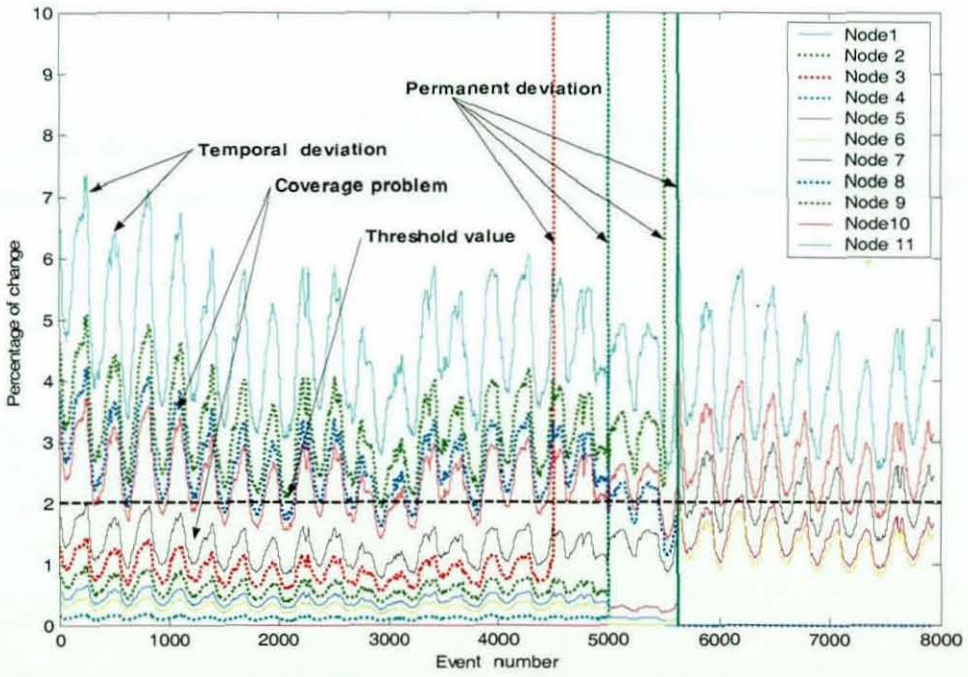

Figure 7-7. Algorithm Detection with Weighted Residual Changes

When the experiments were repeated with an algorithm that removes the deviated faulty nodes after detecting them, the effect of the faulty nodes on collected data accuracy was reduced to only that period of monitoring time before the faulty node was removed and the reading accuracy returned to 100%, as shown at Figure 7-8.1. This happened due to the short period of

time the faulty deviated node occurred before it was isolated, as shown in Figure 7-8.4. However, this improvement in accuracy came as a tradeoff with a reduction in the number of neighbour nodes that the network depends on for collecting data, for collaboration, and for communication. Moreover, comparing Figures 7-8.1 and 7-6.1 shows an instantaneously higher impact on the accuracy of the collected data concerning deviated faulty nodes when the algorithm works to remove suspected nodes because of a reduction in the number of neighbour nodes. This instantaneous change in value depends on the degree of change in readings compared to other readings in the neighbourhood. If the suspected permanent nodes are not isolated, their residual effect on the neighbourhood's data accuracy continues after they become faulty as shown in Figure 7-6.4.

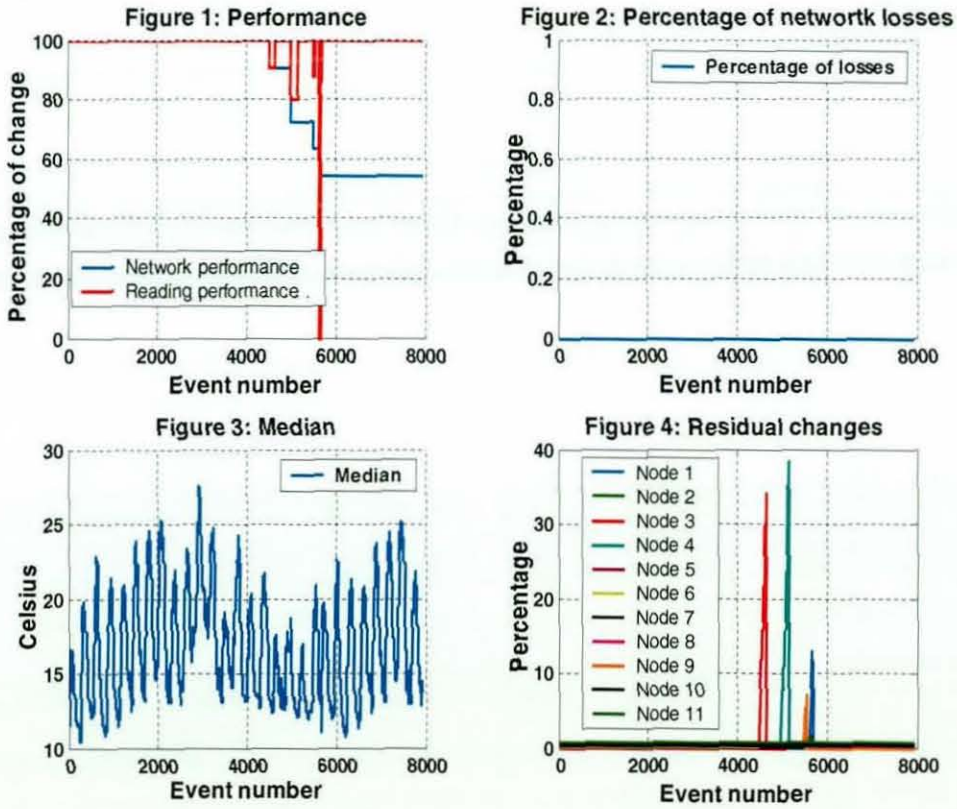

Figure 7-8. Simulation without Loss and with Isolating Deviated Nodes

Injecting the data set with different percentages of random losses shows a fluctuation in readings and in the network's performance, as can be seen from the heavy fluctuation in the data accuracy and in the network's performance, as shown in Figures 7-9.1 and 7-10.1. These fluctuations in data accuracy

depend on the number of neighbour readings received, the status of lost neighbour readings (i.e. whether they are healthy or faulty), and the degree of closeness of the data. Losses affect the median calculation due to variations in the number of healthy and unhealthy nodes per event. Such variations may lead to deviations in median value from the expected value of the phenomenon because there may be a majority of unhealthy nodes at a certain time interval. To overcome this problem, the algorithm was modified so that it would store the last median calculation and compare it with the next median value. If the difference then exceeded the expected change in the phenomenon measurements, it rejected the new calculated median and used the last stored one. By testing the algorithm with this new modification we achieved more stable median calculations and fault detection.

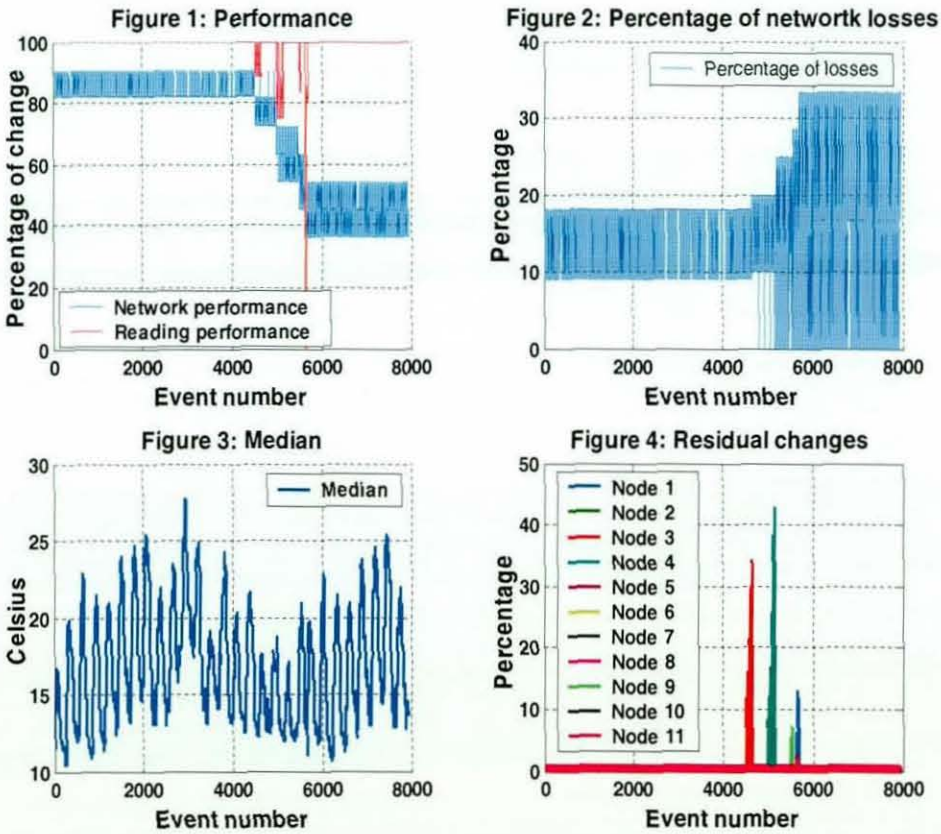

Figure 7-9. 20% Loss Data (Algorithm without a Classifier)

The comparison of the proposed algorithm's detection in experiments with loss and lossless mediums showed a difference in the algorithm's detection time and order. For example, detecting two deviated nodes which occur at the same time depends on the percentage of losses so, when losses were less

than 40%, the detection of the two faults occurred simultaneously. But, when losses increased over 40%, the algorithm detected the two deviated readings at around 200 events apart.

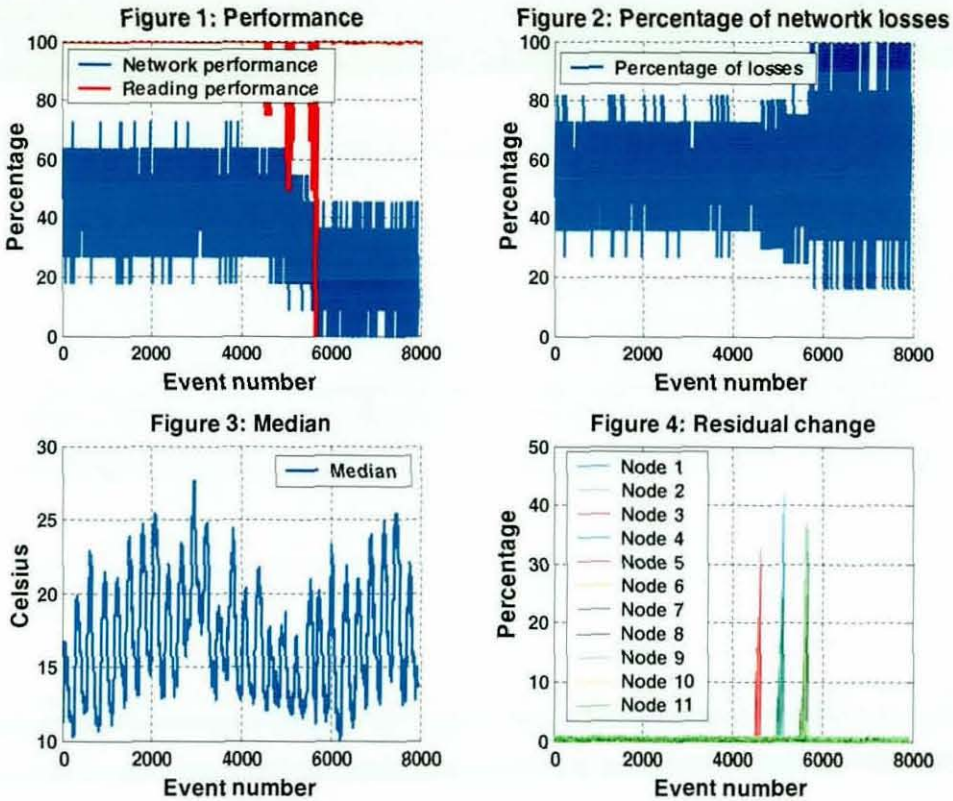

Figure 7-10. 80% Losses Data (Algorithm without a Classifier)

Moreover, these experiments show that increases in the percentage of losses cause the algorithm to detect deviated nodes in a different order than they occurred. This shows clearly that the losses affect the detection and isolation of faulty nodes and may also cause them to be undetected if the calculated weighted residual is less than the detection threshold due to the number of packets received from that neighbour. Because of this, the proposed algorithm was redesigned such that it takes into account in its analysis the percentage of neighbour packet losses. This is done such that the threshold decision of the monitoring window is based on the number of packets received from neighbours not monitoring window size.

The same experiments were conducted again to test the algorithm detection when a classifier was included in its function; the classifier used in the

experiments is discussed in Chapter 4. These experiments showed that the algorithm with a classifier took more time to isolate the faulty deviated nodes if more than one fault occurred at a time. In such cases, the algorithm classified them depending on their values as the environment or the phenomenon changed. Due to this, the deviated nodes were found to have a higher impact on the data's accuracy when the algorithm with a classifier was used. This can be seen if Figures 7-8.1, 7-9.1 and 7-10.1 are compared with 7-11.1, 7-12.1 and 7-13.1.

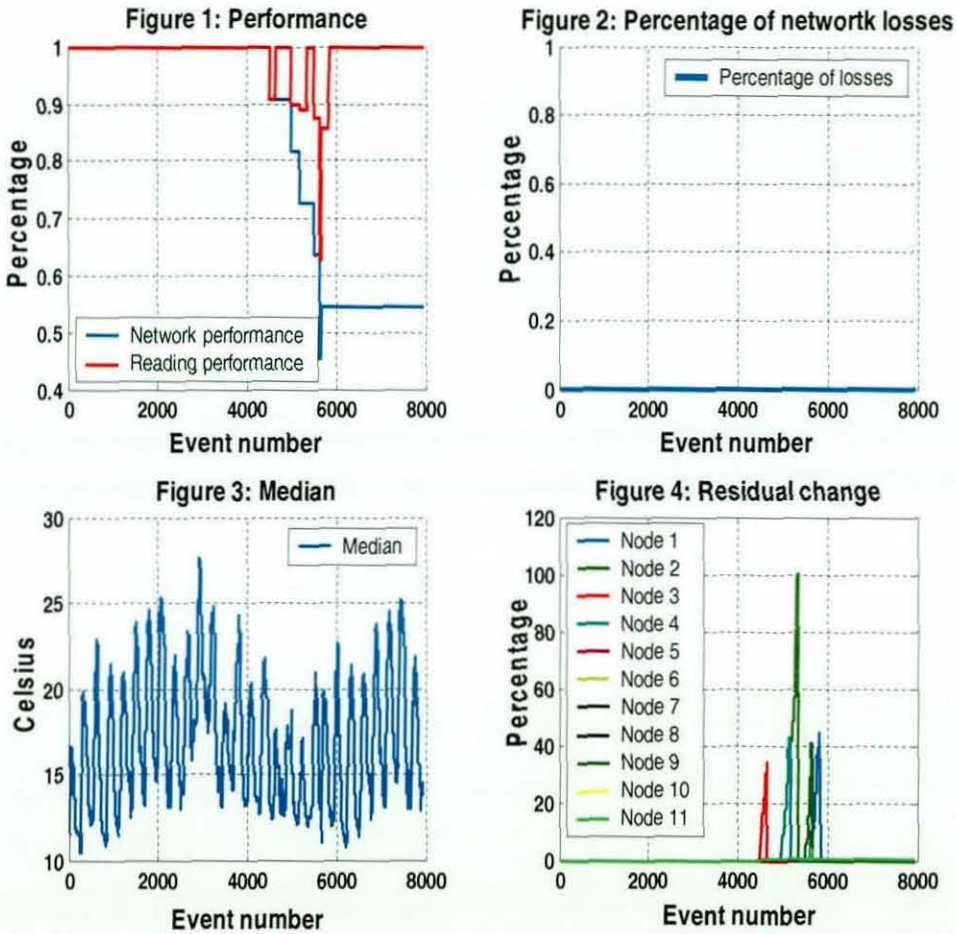

Figure 7-11. Data without Loss (Algorithm with a Classifier)

Finally, Figure 7-14 shows the warning packets released by the algorithm with and without a classifier at 50% losses without using the message exchange module. The figure shows almost the same number of sent messages but these released messages are different in terms of time because the algorithm with a classifier took longer to make the detection than the one that had no classifier.

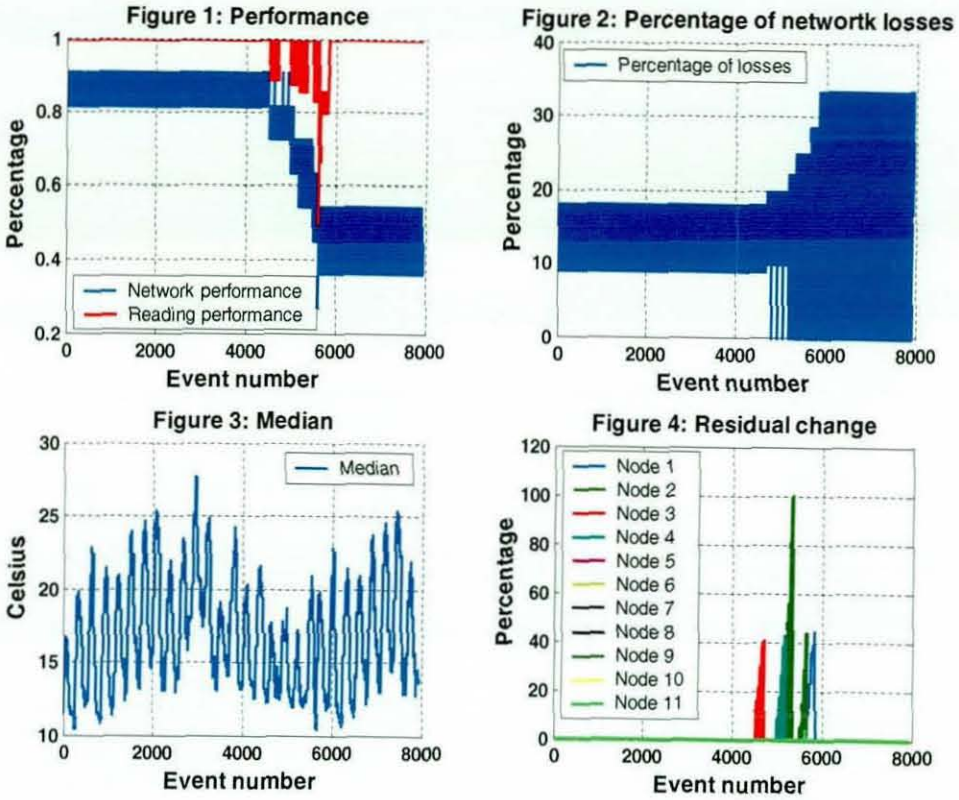

Figure 7-12. 20% Data Loss (Algorithm with a Classifier)

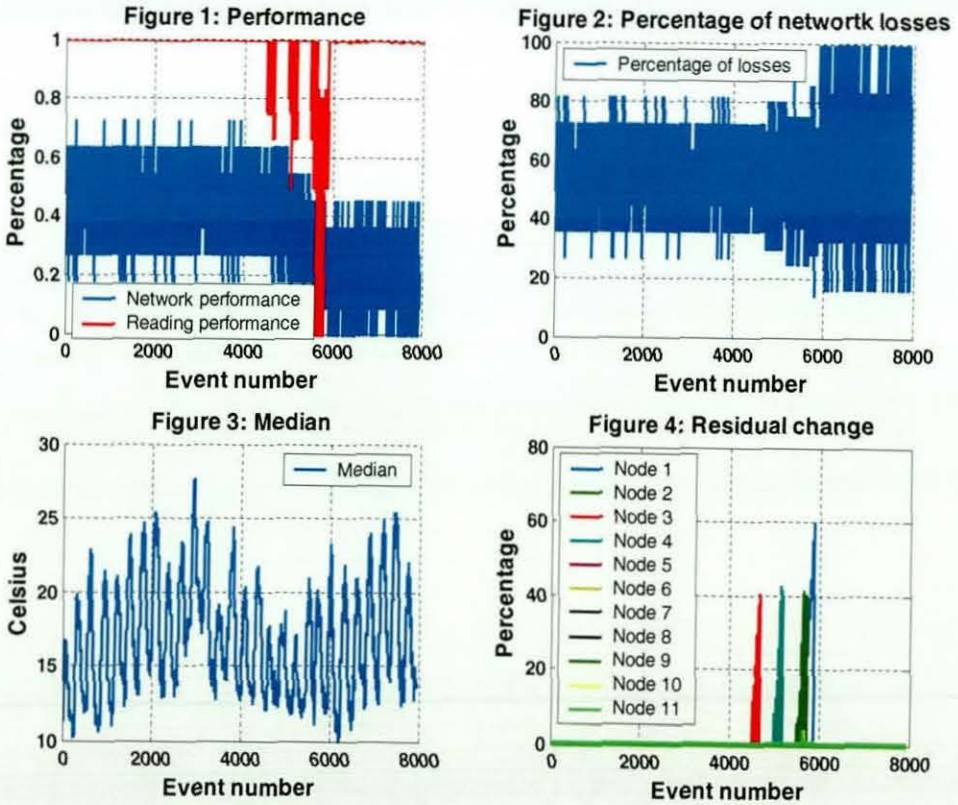

Figure 7-13. 80% Loss (Algorithm with a Classifier)

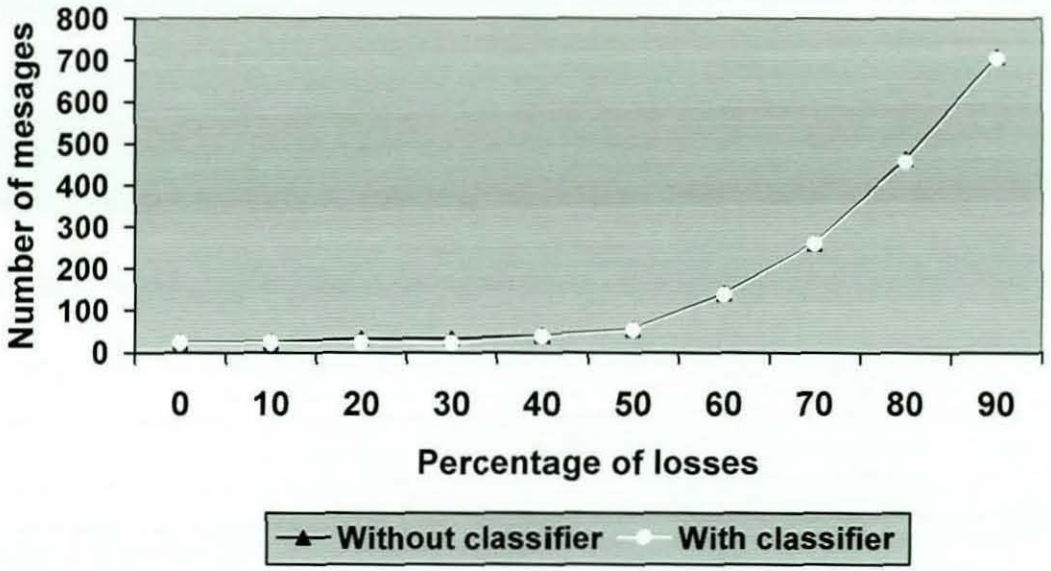

Figure 7-14. Number of Messages Sent for Algorithm With and Without a Classifier

7.4 Results For Real World Data Sets

These sets of data consisted of two (indoor and outdoor) real world application experiments (i.e. Intel LAB experiment [57], and UC Berkley Botanic Garden [58] respectively). Each of these data sets had its own characteristics which were tested using statistical analysis (described in [50], [51] and in Chapter 6) to detect the location and values of outliers before simulating them with the algorithm. This analysis was then used to test the detection of the algorithm under different circumstances and the effect of data losses on it. This is to evaluate the performance of the algorithm's detection of both spatial and temporal change and the impact of different real world scenarios.

All testing scenarios started with a sample data set that had no faulty sensors in order to ensure that implementing a modification to the algorithm did not affect its functionality. Then modified algorithm codes were tested with different data sets; the number of detected deviations and their time of occurrence were then compared with results from using statistical methods. Finally, the undetected deviations were studied, the algorithm code was modified to overcome the problems and the experiment was repeated again.

All the experiments showed a good level of compatibility with statistical results.

7.4.1 Intel LAB Data

The same three metrics were used in Section 7.3.2 along with two additional metrics. The first calculates the number of warning messages released by the algorithm per size of monitoring window. This shows the effect of window thresholds on changes in the warning packets released by the algorithm. Since there is no ground truth for the measured phenomenon, the metric which shows the percentage of detected faults was used to indicate the accuracy of the algorithm's detection. This metric is the ratio of the common outliers detected by the algorithm and the Box-Whisker method, as opposed to the total number of outliers detected by Box-Whisker.

Applying the Box-Whisker method to neighbour measurements of Node 1 to isolate outliers gave the results recorded in Table 7-2 below.

Data set	Total deviations	Low limit C^o	High limit C^o
38% losses	160331	8.4	41.7
65% losses	86246	9.2	39.9

Table 7-2. Box-Whisker Method Output of Intel lab Data Set

Implementing the algorithm for this data set shows that the algorithm detected 101923 changes of value for all nodes, as outliers confirmed 64857 to be faulty deviations for a data set with 38% losses. When 108133 were detected, outliers confirmed 83891 of them as fault deviations with a data set with a 65% loss. A comparison of these detections using the Box-Whisker method showed that, with a data set of 38% losses, the algorithm detected 36% fewer outliers. This is due to similar deviation changes in undetected data that reduced their weighted residual on the accuracy of data collected in the network. With a data set of 65% losses, the algorithm detected 97% of the outliers detected by the Box-Whisker method.

Figure 7-15 shows the neighbourhood data accuracy and the network functionality of Node 1 without removing suspected nodes. The figure also shows fluctuations in the accuracy of the collected data and the network's performance as a result of the residual impact of deviated data on the neighbourhood data accuracy, as shown in Figure 7-15.4. This continues up to the time where the fluctuations become very heavy due to the effect of losses and the preponderance of unhealthy readings as they become the majority (i.e. as shown in Figure 7-15.1). Afterwards, this heavy fluctuation becomes constant when the number of permanently deviated nodes is greater than the healthy nodes at the end of the experiment (i.e. from event 680000 upwards). Normally, this heavy fluctuation does not occur in *WSNs* due to the redundancy that schedules the function of nodes, which makes the probability low of failure occurring at the same time. Even if this happens, it can be detected from the dramatic change in data accuracy and the increase in the weighted residual which moves up to a constant level for a long period as shown in Figure 7-15.1.

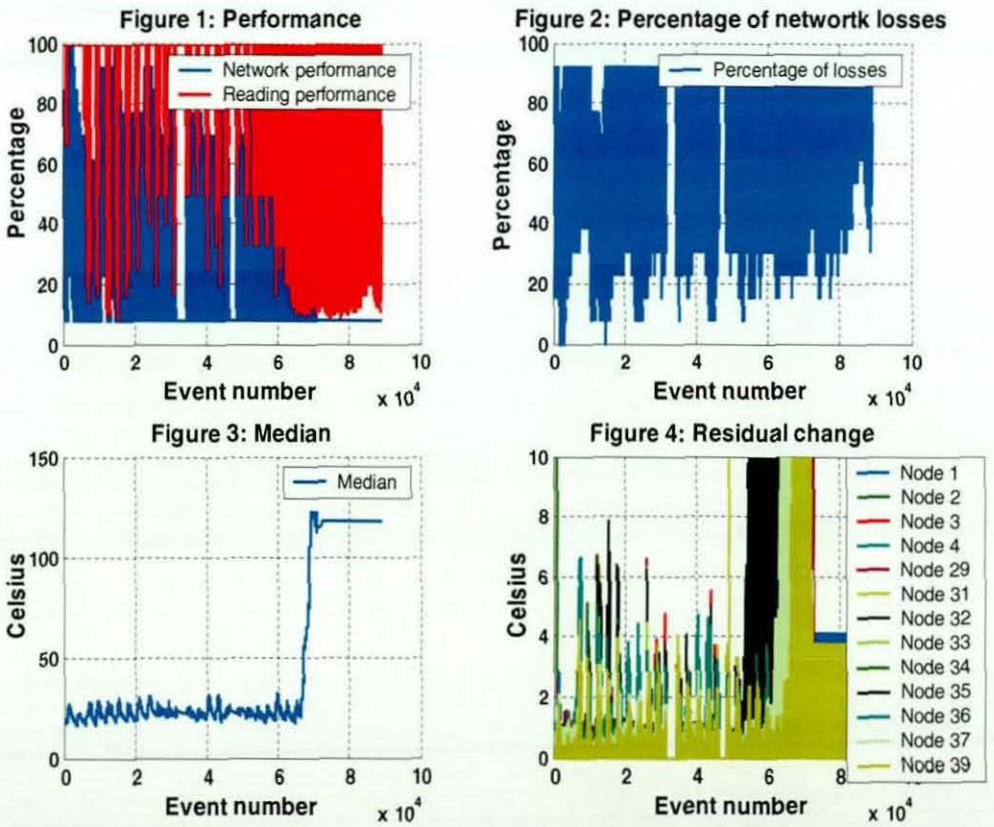

Figure 7-15. Intel Lab Data and the Algorithm without removing deviated nodes

The figure also shows that even when there is a heavy fluctuation in neighbour readings due to losses, the calculated median does not deviate until permanently faulty nodes become the majority (as shown in Figure 7-15.3). This is due to using the new calculated median and the old stored median values and comparing the differences with the application's permitted degree of change.

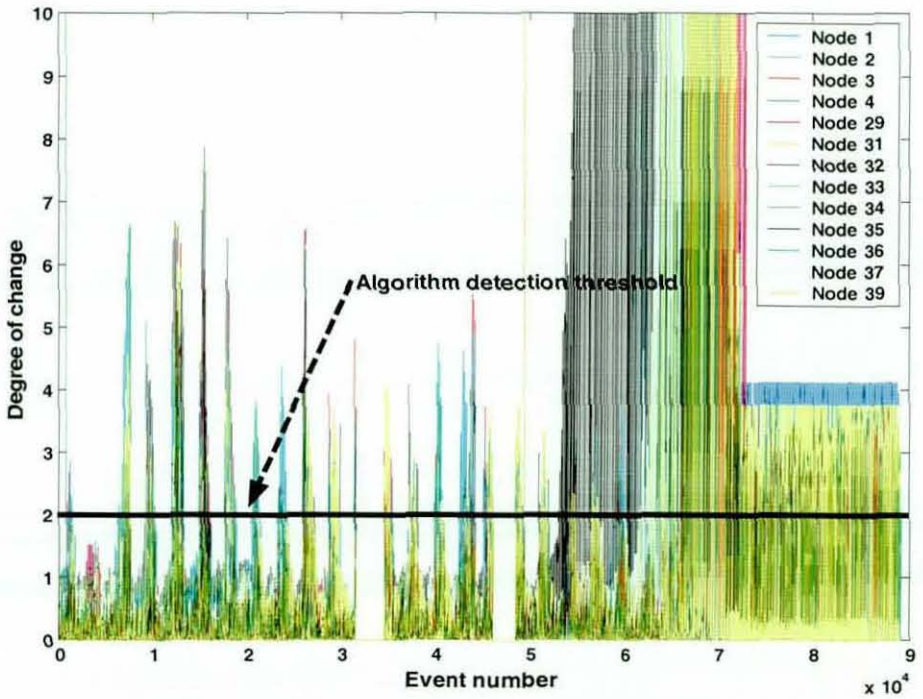

Figure 7-16. Threshold Based on Constant Value

Figure 7-16 illustrates the proposed algorithm's calculation of the weighted residual for individual node measurements with respect to the neighbourhood median. If this figure is compared with Figure 7-15.4, almost the same changes in detection can be seen but with different residual values; these depend on the number of deviated readings at each time interval.

If the algorithm is allowed to isolate faulty deviated nodes at a specified monitoring window, the neighbourhood's performance is improved but with any new deviated node making a higher impact, as shown in Figure 7-17.1. This happens because of the increase in the impact of the residual on the collected data as a result of reducing the number of data samples at each time interval. On the other hand, not removing the diverted reading affects the

accuracy of the collected data for the period it for which occurs, as shown in Figure 7-15.1.

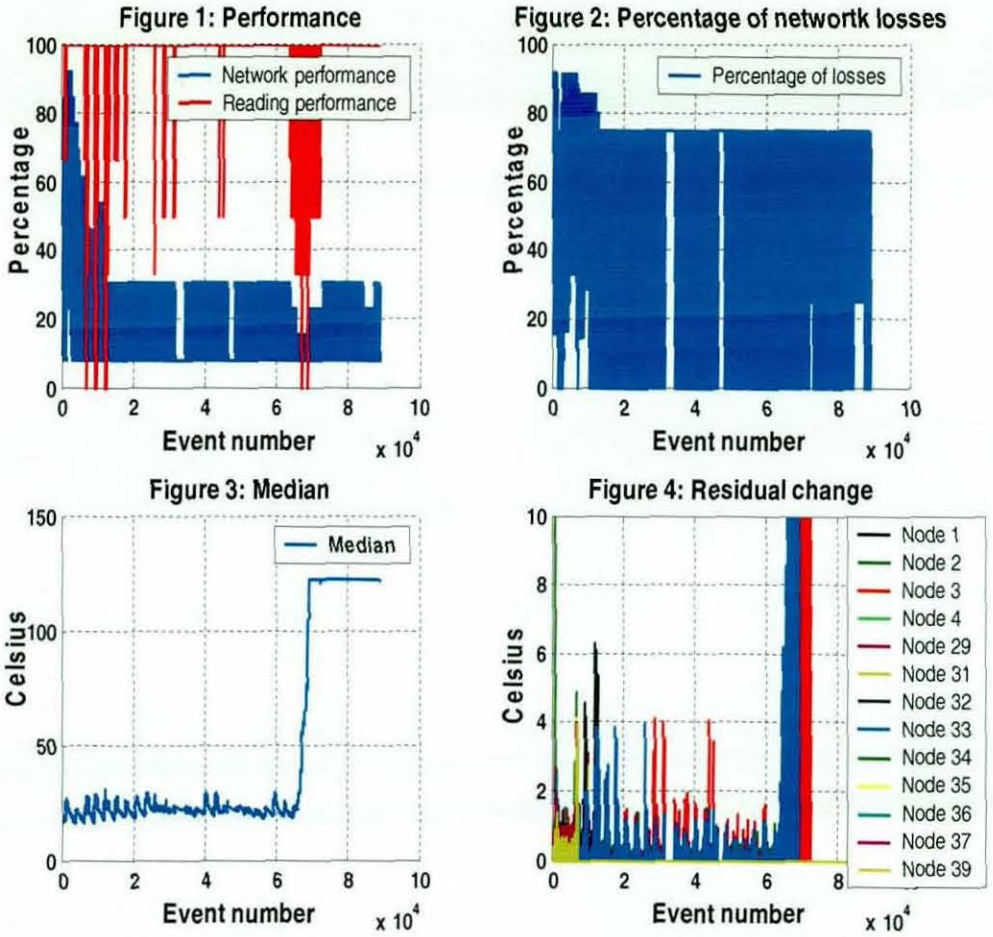

Figure 7-17. Intel Lab Data and the Algorithm with removing deviated nodes

Figure 7-18 shows the proposed algorithm's weighted residual calculation when it is allowed to isolate faulty deviated readings. The figure shows clearly the low number of deviated nodes and the degree of their effect if compared with Figure 7-16.

Figures 7-19 and 7-20 illustrate a comparison between the accuracy of readings and network performance with and without removing faulty nodes. Figure 7-19 shows that removing the deviated nodes improved the accuracy of the data but Figure 7-20 shows a reduction in the network's performance due to this removal because of the reduction in the number of nodes in the neighbourhood (network performance depends on connectivity and collected data).

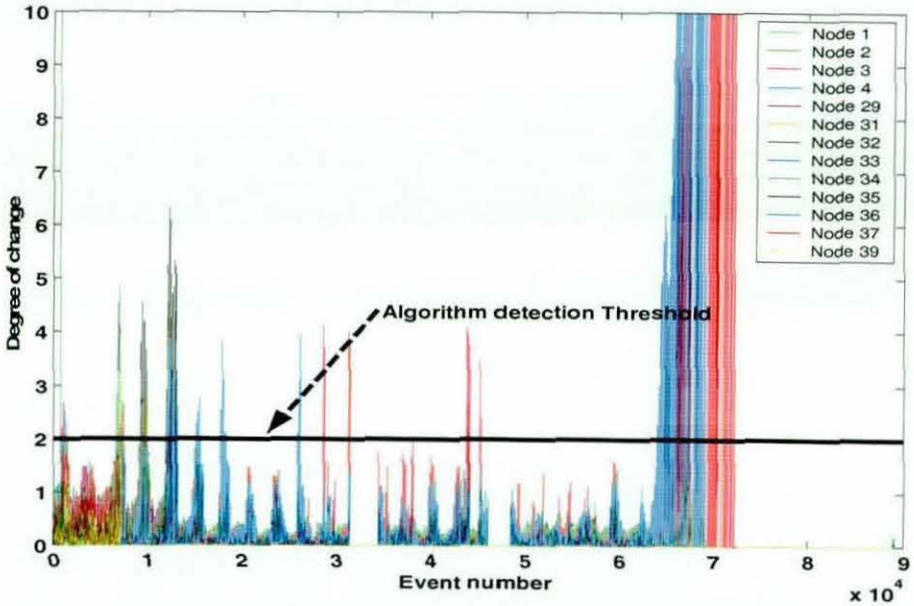

Figure 7-18. Threshold Based on Constant Value

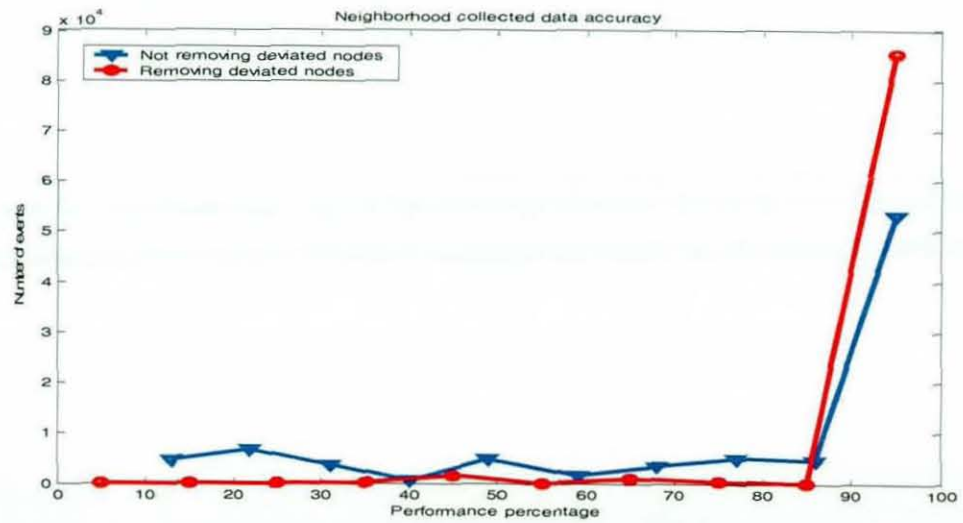

Figure 7-19. Reading Performance

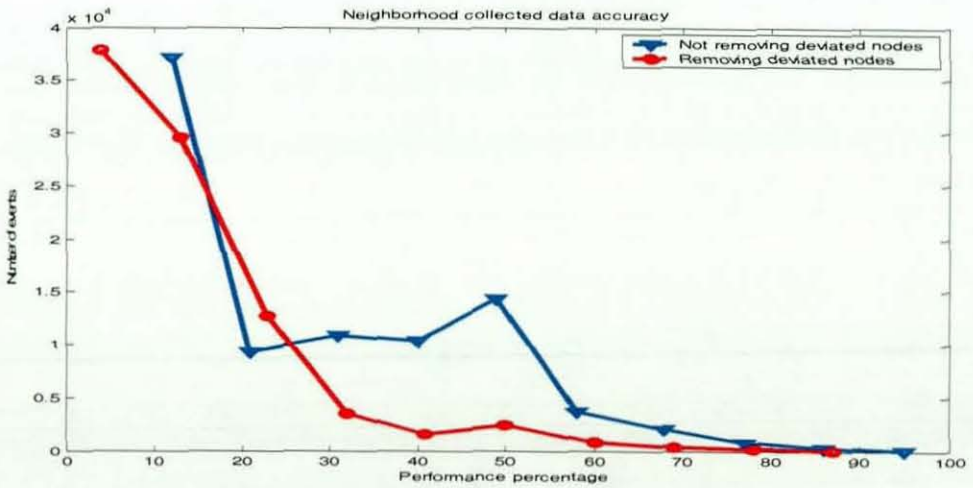

Figure 7-20. Network Performance

Several experiments were conducted to check the possibility of predicting the lifetime of nodes from losses or measurement changes by comparing these with other neighbours. An analysis of data collected from other research showed that, as voltage levels reduce, losses increase, while measuring levels increase or decrease dramatically as in [87]. The problem is that losses in *WSNs* are not only caused by power depletion. This makes any prediction more difficult as it increases the complexity of the proposed algorithm, requires more memory, and needs increased power consumption. This type of analysis can be carried out at the sink but most applications do not send individual node measurements and, if they do, these may be lost in a multi-hop system. (This point could be a focus of future research in order to find parameters that can deduce a node's lifetime.)

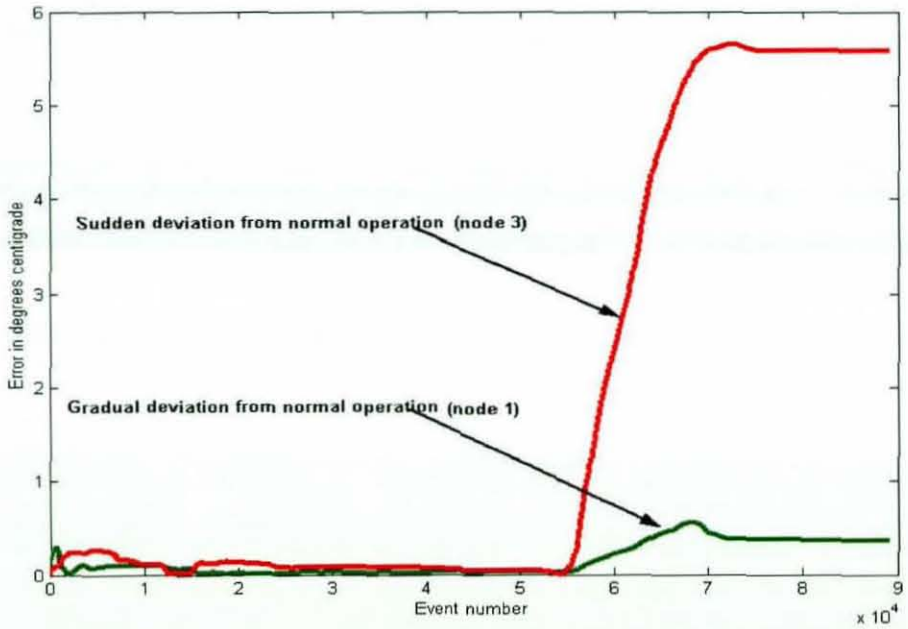

Figure 7-21. Absolute Moving Average Residual Values of Nodes for the Intel Lab Experiment Data Set

Moreover, testing the behaviour of faults in *WSNs* by using a moving residual average between median and readings indicated two behaviour faults, as shown in Figure 7-21. The first occurred suddenly and without any previous indication, such as demonstrated by Node 3 in Figure 7-21 while the second showed slow deviated values over time, as illustrated by Node 1 in Figure 7-21. However, each of these occurred alongside an increase in deviated node packet losses as if this was associated with power. As a result, the algorithm's

detection confidence, as explained in Chapter 4, uses event tests and does not depend on statistical models.

Table 7-3 shows the difference between the algorithm's performance with and without a classifier in terms of the number of times detection occurred before faulty nodes were removed from Node 1 neighbours. The algorithm without a classifier sent 227 faulty warnings and removed 11 nodes; it also sent 7534 suspected dead warnings. The algorithm with a classifier, on the other hand, sent 304 faulty warnings and removed 11 nodes and sent 7838 suspected dead warnings. Moreover, the table shows that the algorithm with a classifier achieved a greater number of deviation detections during the experiments because this algorithm considered the similar changes between two readings as a change in the environment or the phenomenon. This reduces the number of detections in some of the small windows below its threshold; it does not satisfy the threshold of a big window.

Node		1	2	3	4	29	31	32	33	34	35	36	37	39
Algorithm without classifier	Detection event	removed	Not removed	66240	60960	12480	10080	15360	15840	44160	53760	7200	62880	66720
	Number of messages until remove	3	3	17	15	14	12	17	15	38	37	7	20	18
Algorithm with classifier	Detection event	66240	Not removed	66240	60960	12960	10080	15840	15840	44160	53760	7200	62880	Not removed
	Number of messages until remove	20	3	11	13	12	11	19	13	33	35	6	17	100

Table 7-3. Detected Faults and Removed Nodes for the Algorithm Operating on Node 1

To check the effect of selecting neighbours on the algorithm's operation, 8 nodes were selected to be neighbours of Node 1 instead of 11, depending on

the level of correlation with Node 1 measurements. These experiments showed a slight improvement in data accuracy and in network performance.

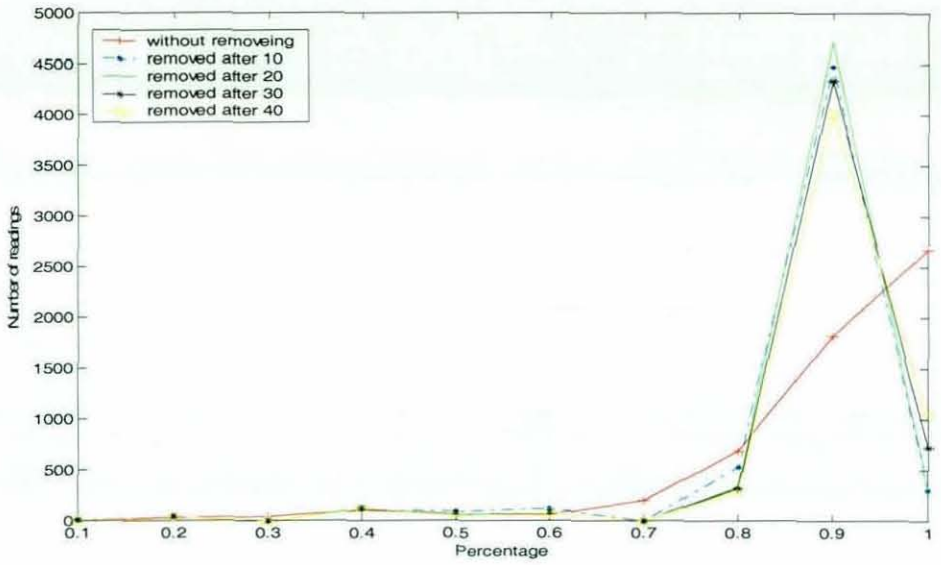

Figure 7-22. Network performance for Selected Neighbours

Some experiments were conducted to test the effect of the number of fault detections and the removal of deviated nodes on network performance and data accuracy. Figures 7-22 and 7-23 present a number of events, together with network performance and data accuracy, for different numbers of detection before removing the suspected node. The figures show improvement between removing and not removing suspected nodes, but with different levels of detection the improvement was small. This is because detection depends on the type of fault (i.e. permanent or temporary) and the threshold of the monitoring window.

Figure 7-24 outlines the relationship between the number of messages sent by the algorithm and the percentage of losses. It indicates clearly that as the percentage of losses increases, the number of warning messages also increases because most of these warning messages relate to suspected dead nodes. This high rate of warning messages can be solved by adding a message module to the algorithm that will track the messages released from it and decide if there is a problem or when to stop sending the message. This is discussed in Chapter 4.

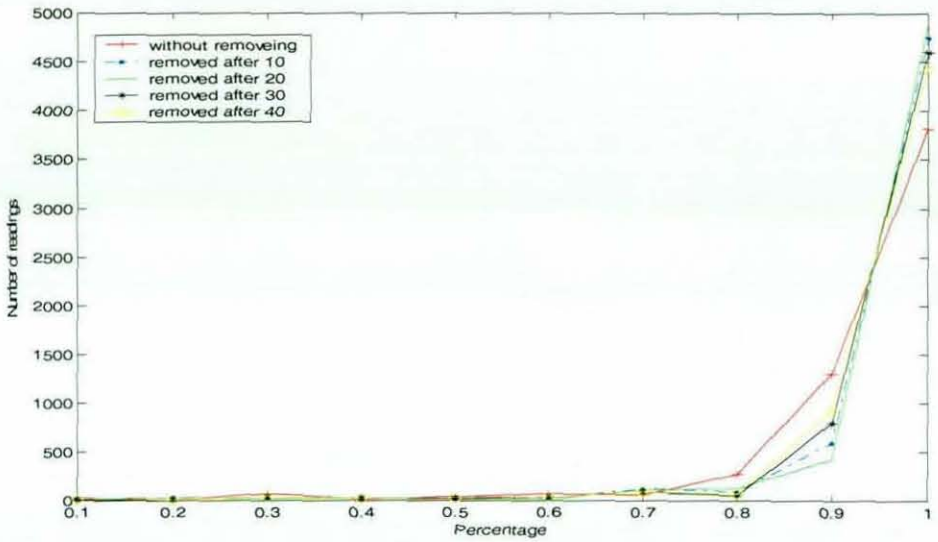

Figure 7-23. Accuracy of Data Collected by Node

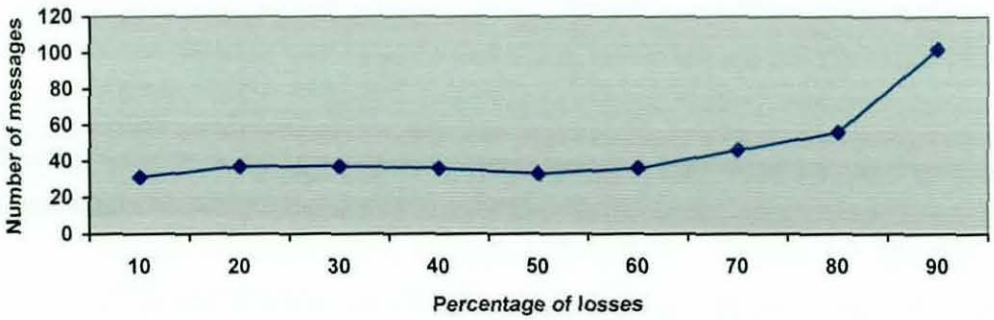

Figure 7-24. Number of Messages versus the Percentage of Losses in 6 Samples with a Small Window and 24 Samples with a Big Window with 40% Threshold

	1	3	4	33	35	37	39
Node1	--	66240	60960	15840	53760	62880	66720
Times	--	17	15	15	37	20	18
Node2	--	66240	60960	18240	24000	24000	66720
Times	--	13	6	29	21	16	24

Table 7-4. Event Number of Removed Nodes

These experiments were repeated so that the analysis could be carried out on Node 2 and its neighbours. Table 7-4 shows the detection interval and the number of messages sent by Nodes 1 and 2 before isolating faulty nodes. As can be seen from the table, some of the nodes detected faulty nodes at the same event while others detected them at different times. The table shows

that more messages were sent by Node 2 because of different neighbour packet losses.

7.4.2 Botanic Garden Data Set

This data set has a low level of losses, was a single-hop network and had very few outliers (as discussed in Chapter 2 and shown in Table 7-5). However, it also contains temporary changes as a result of the impact of outdoor environmental changes on the sensor nodes and has different coverage due to the altitude of each group of sensors.

Data set	Box-Whisker method			Normal distribution		
	Total outliers	Low limit	High limit	Total outliers	Low limit	High limit
0% losses	19	3.5	30.7	2771	11.4	26
14% losses	19	3.5	30.7	2695	11.4	26

Table 7-5. Box-Whisker Method Output for the Botanic Garden Experiment Data Set

The same metrics as in Section 7-4.1 were used but the Box-Whisker method was replaced with the Gaussian PDF limits test because this data set does not have outliers; instead, it records changes in data due to changes in network nodes coverage during the day.

The experiments were conducted to test the degree of impact of these temporary changes on the accuracy of readings and the network's functionality. Moreover, they tested the effect of threshold level and window size on the proposed algorithm's detection.

Implementing the algorithm shows that it detected 2979 changes from all nodes and confirmed 2797 as faulty deviations at 0%packet loss. While with 14% losses the algorithm detected 2882 changes and confirmed 2752 as faulty deviations. These detected deviations were due to temporary changes in sensor measurements as a result of different levels of coverage and changes in environmental conditions during the day. Comparing these levels of detection with detection using the normal distribution limits method [96]

shows that the difference between the algorithm's detection and normal distribution limits was 3%.

Figures 7-25 and 7-26 illustrate changes in Node 1 neighbours and the effects of these changes on collected data accuracy, network performance and residual changes with different sizes of monitoring window. Moreover, they show the calculated median of the neighbourhood.

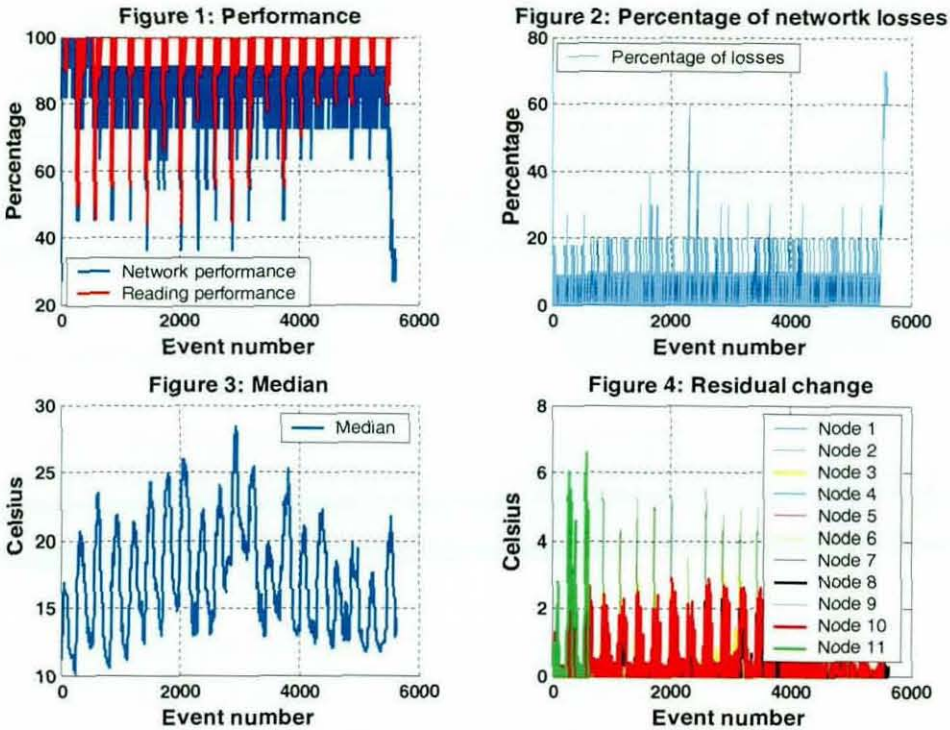

Figure 7-25. The Algorithm's Operation with a 50 Sample Window Size (i.e. 5 Small Monitoring Window Sizes, each with 10 Samples) and a 2 Degree Centigrade Threshold

Figure 7-25.1 illustrates the effect of temporary changes on the accuracy of the data and the network's performance using a 50 sample monitoring window size (i.e. 5 small monitoring window sizes, each with 10 samples) and a 2 degree centigrade change in the data threshold. This figure shows that there is an instantaneous degradation of both network performance and reading accuracy due to these temporary changes in Node 11 (as shown in the algorithm's weighted residual detection in Figure 7-26). After the temporarily deviated node has been detected and isolated by the algorithm, reading accuracy was improved, as shown in Figure 7-25.1.

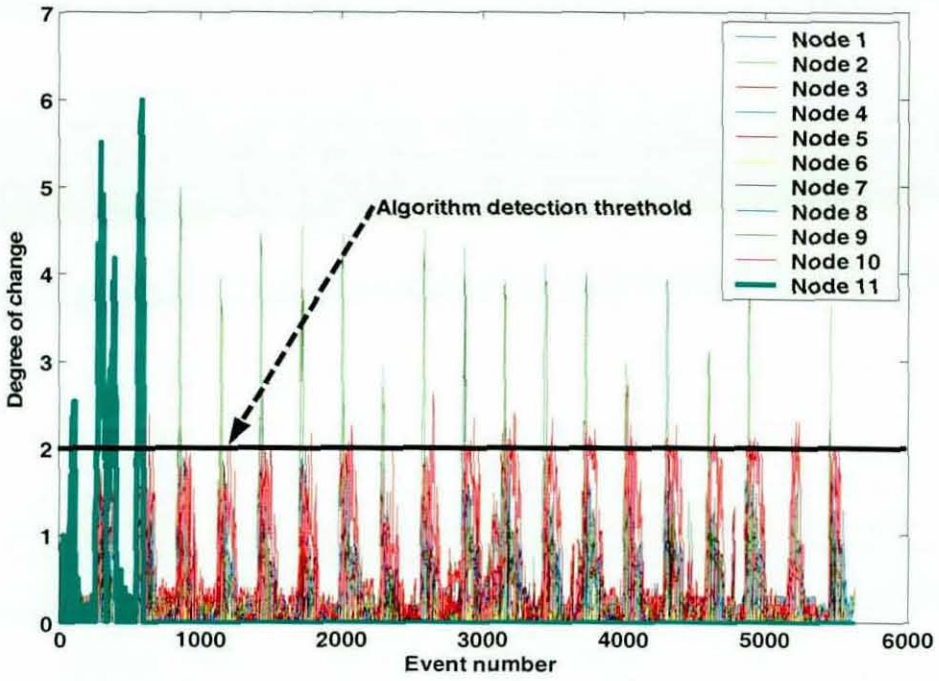

Figure 7-26. The Algorithm's Operation with a 50 Sample Window Size (i.e. 5 Small Monitoring Window Sizes, each with 10 Samples) and a 2 Degree Centigrade Threshold

This detected temporary change occurred only for a small period of time and, in order to detect it, the monitoring window and its threshold must be small. When the size of the small monitoring window was reduced from 10 to 5 samples, the algorithm detected 3 deviated nodes, as shown in Figure 7-27. If the detection threshold is then increased from 2 to 5 degrees centigrade with a 50 sample window size (i.e. 5 small monitoring window sizes, each with 10 samples), the algorithm did not detect any deviation. This is shown in Figure 7-28.

These experiments showed that, in order to detect temporary changes of coverage or in the environment, small and large window sizes, together with their thresholds, should be selected depending on network tolerance to this duration and value change. Because of this, the goals of the network application, (such as the deployment goals, the accuracy of the collected data and network users) all play a role in determining the monitoring window sizes and the threshold levels.

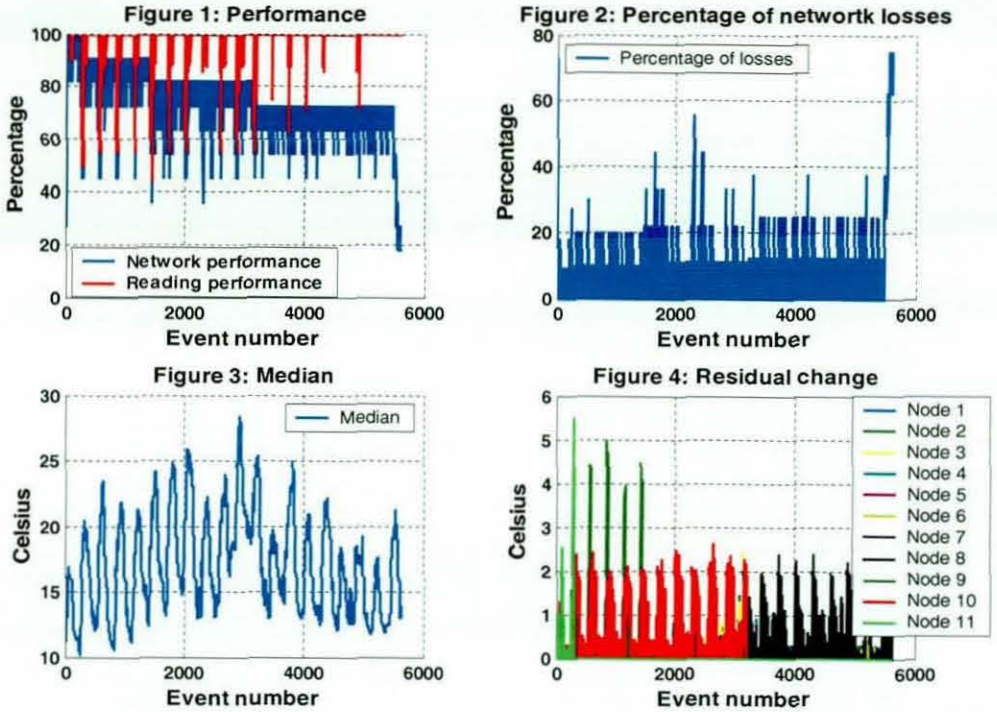

Figure 7-27. The Algorithm's Operation with a Small Window of Size 10, A Large Window of Size 5 and a Threshold of 2 Degrees Centigrade

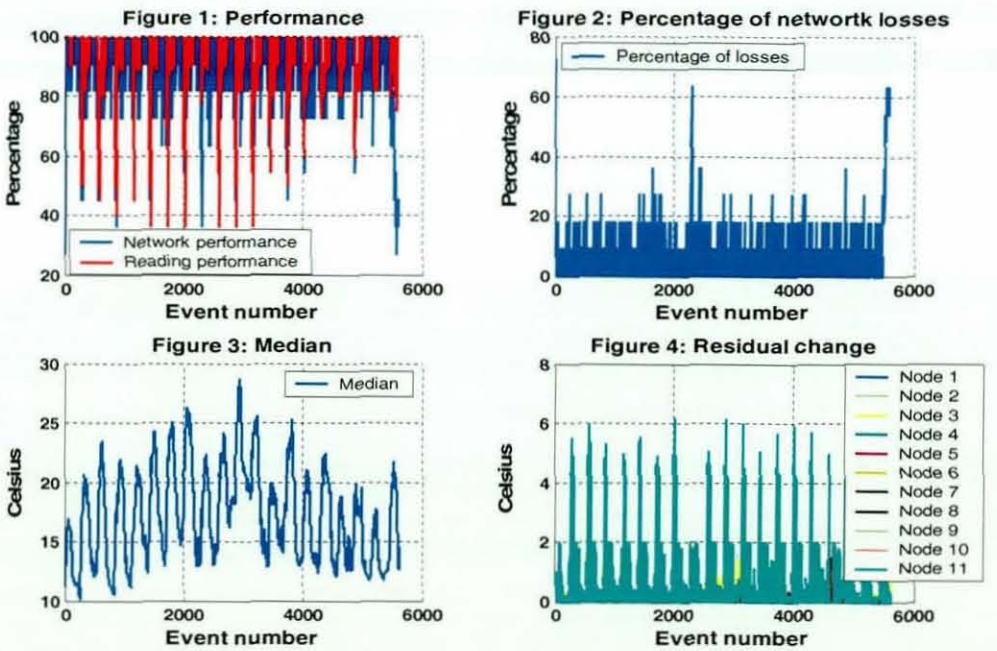

Figure 7-28. The Algorithm's Operation with a 50 Sample Window Size (i.e. 5 Small Monitoring Window Sizes, each with 10 Samples) and a 5 Degree Centigrade Threshold

The experiments were repeated for the proposed algorithm with a classifier and these showed that the behaviour of the algorithm with and without a

classifier was almost the same. Both detected Nodes 11, 9 and 10 at event number 120, 288 and 3768 respectively. However, these experiments also show a difference in the average number of warnings, as illustrated in Table 7-6.

	With loss	Without loss
With classifier	123	81
Without classifier	104	57

Table 7-6. Average Number of Messages Sent to the Sink

The algorithm with a classifier showed a better ability to distinguish between changes that occurred due to faults and changes that happened as a result of the impact of different environmental and external factors. This is because the algorithm with a classifier compares the reference median for each node to another and if the comparison shows a common degree of change, it considers this as a normal change.

To check the effect of threshold and window size on the release of warning packets, several experiments were conducted. Figure 7-29 plots the relation between the number of messages released from the algorithm versus large window size (i.e. a multiple of 5 samples) for a fixed window threshold. This shows that, as window size increases, the number of warning packets stays constant up to a certain value; then it starts to increase. This is due to the occurrence of the detection some time before the deviated node is isolated as a result of not satisfying the threshold value of the large monitoring window. On the other hand, Figure 7-30 shows the relation between the number of messages versus the detection threshold of a large window of size 50. It illustrates that, as the detection thresholds increase, the released messages are almost the same until there is a sudden drop (i.e. over 14 samples in the experiment) because the number of changes with this window is lower than the window threshold. This illustrate that the monitoring window size and its threshold should set depend on the application required fault tolerance.

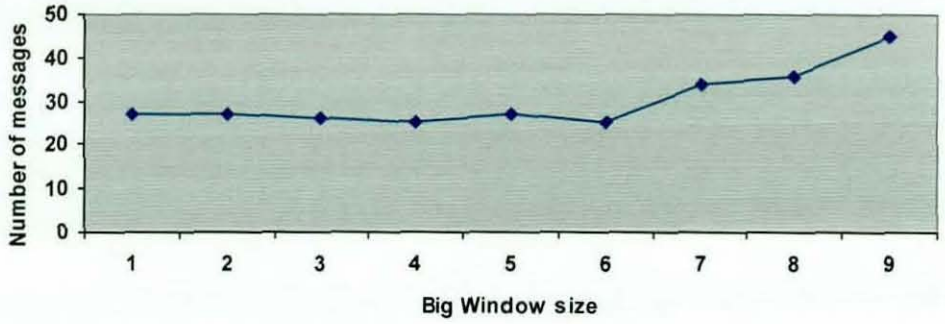

Figure 7-29. Number of Messages with 5 Samples for a Small Window Size and 40% Threshold

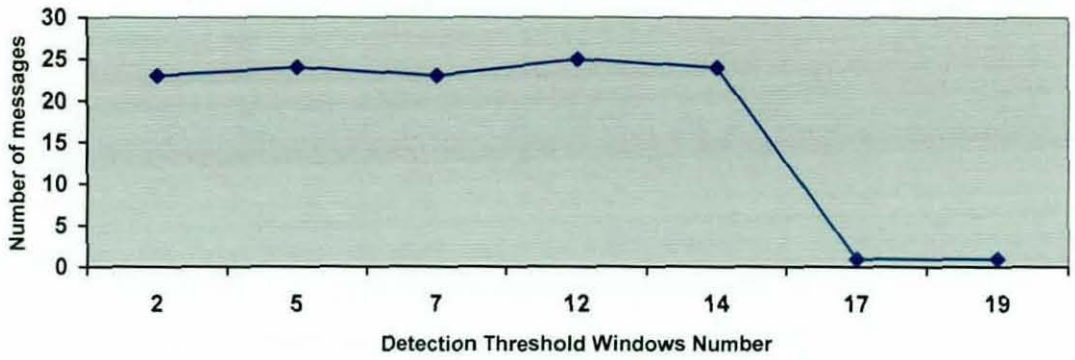

Figure 7-30. Number of Messages verses Threshold for a Size 50 Sample Window

7.5 Reduction of Power Consumption in the Algorithm

Although the proposed algorithm does not consume excessive power due to the usage of common information at the node and its dependency on processing (as discussed in Chapter 6), it still uses RAM memory, and any reduction in is useful. Savings can be made either by selecting a node from the neighborhood to do the test every period, or by applying a particular condition so that the algorithm's analysis starts when the deviation is detected.

7.5.1 Algorithm location

One way to use the algorithm is in a fully distributed scheme at every node; this allows each node to be involved in monitoring its neighbours. However, due to this there will be redundant analysis and more energy will be consumed for receiving and storing the reading. Many applications have this

facility, such as raw data collection applications for environmental measurements, tracking applications, and detecting applications. These types of application require continuous listening of neighbours for collaboration tracking, rote selection and detection of the phenomenon especially in abnormal circumstances. This means that the algorithm will not add further energy consumption other than for its computation and the storage of the results if required (unless a warning message needs to be sent) due to the availability of the neighbour readings at the node RAM for collaboration, fusion and aggregation, and for transportation functions as discussed in Chapter 6.

Another way of using the algorithm is in a centralised scheme in a cluster head, with a TDMA/CDMA node controller or with aggregation/fusion nodes. By using this architecture, the cluster head node should collect and gather information. This can save from 40% to 50% of the algorithm's energy consumption, as explained in Chapter 5. (Please note that this method will increase energy consumption at the cluster head) However, the monitoring node should communicate with the new cluster head as the information packet contains the event analysis for each node in the neighbourhood so that it can continue the calculation from the point at which the other node stops. Another solution is to adjust the size of the monitoring window to be the same as that of the cluster head node selected. However, if the cluster node has a problem it may affect the algorithm's results for the period of time it is functioning; this is also true for other application protocols.

Finally, there is an option to implement the algorithm at the sink with more complex calculations after collecting the raw data from the network. This will make the sink a bottleneck and increase the traffic exchange in the network, thus increasing response time.

7.5.2 Random Monitoring Time

This is used for distributed monitoring where each node picks a random time depending on the status and number of the monitored neighbours. This

approach can be used for applications whose protocols are not dependent on storing or listening to neighbour readings such as real time applications (i.e. audio or video transmission). The method starts checking the status of nodes at a random time after the algorithm begins and depends on the percentage of deviation from the calculated median of neighbour readings. It starts by randomly selecting a time for each node and, depending on the closeness between its reading and its neighbours' readings, will calculate the next measuring time. This will reduce analysis but, as a tradeoff, any increase in faults impacts on the accuracy of the neighbour readings.

As shown in Figures 7-31 and 7-32, the algorithm working efficiently depends on the variation in the accuracy of different readings (which vary from one node to another) because it refers to the closeness of the monitored readings with reference to its own readings, not to the median. Furthermore, the monitoring time, the size of the monitoring window, and the detection threshold can be dynamically adjusted, depending on the magnitude and length of the detected fault.

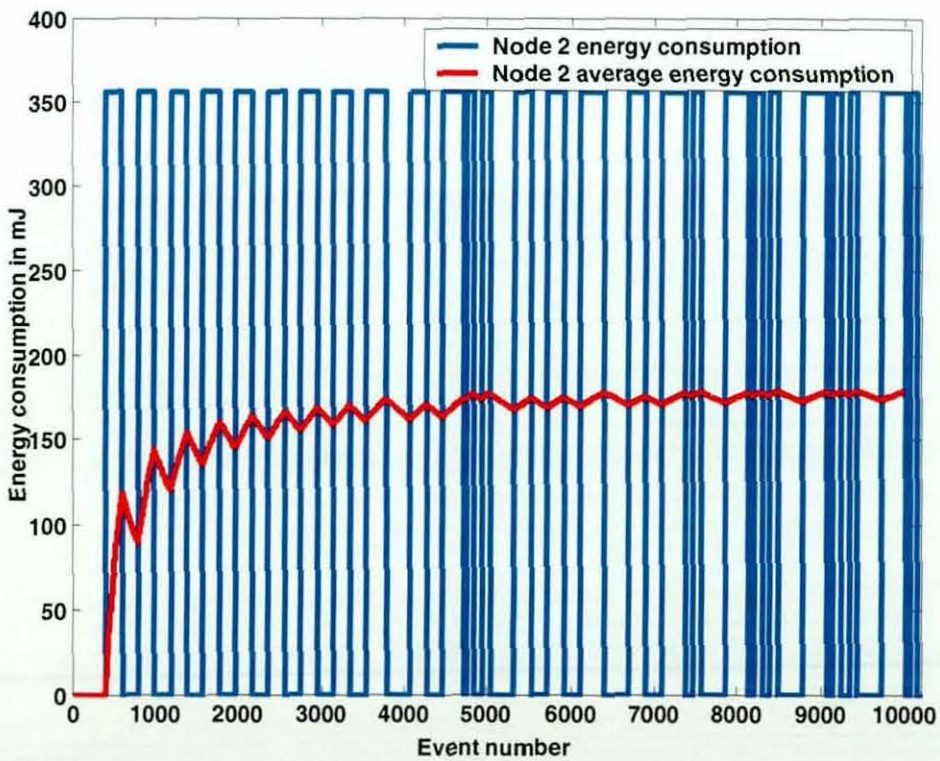

Figure 7-31. Energy Consumption of node 2 using the Algorithm with Random Monitoring Times

When applying the proposed algorithm to make savings in power at node 1 in the Intel Lab data set with a 65% loss, it was found that there was a 15% power reduction when using the algorithm during the 89097 events. While, with the Botanic garden data set with 0% losses, the reduction was 38% during the 5482 events. However, this comes with a detection accuracy tradeoff as it affects the accuracy of the neighbours' fault readings and takes more time in detecting the deviation. Figures 7-33 to 7-34 show the result of running the algorithm continuously and of running the modified version that uses power saving. They show that there is a degradation in reading accuracy and in the network's performance which reached 0% in some instances. This is clearly shown by the residual level in Figure 7-34.4 if compared with that in 7-33.4.

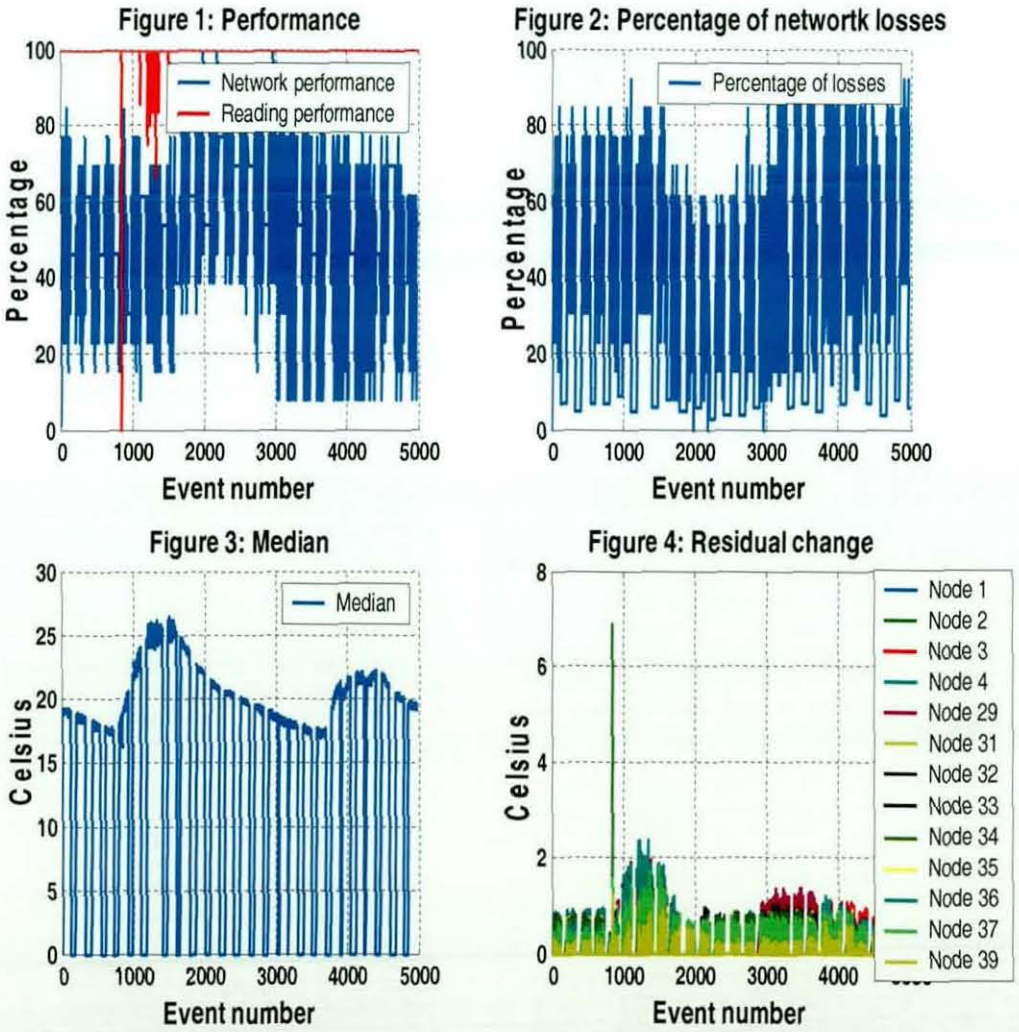

Figure 7-32. The Algorithm Operation Using Power Saving

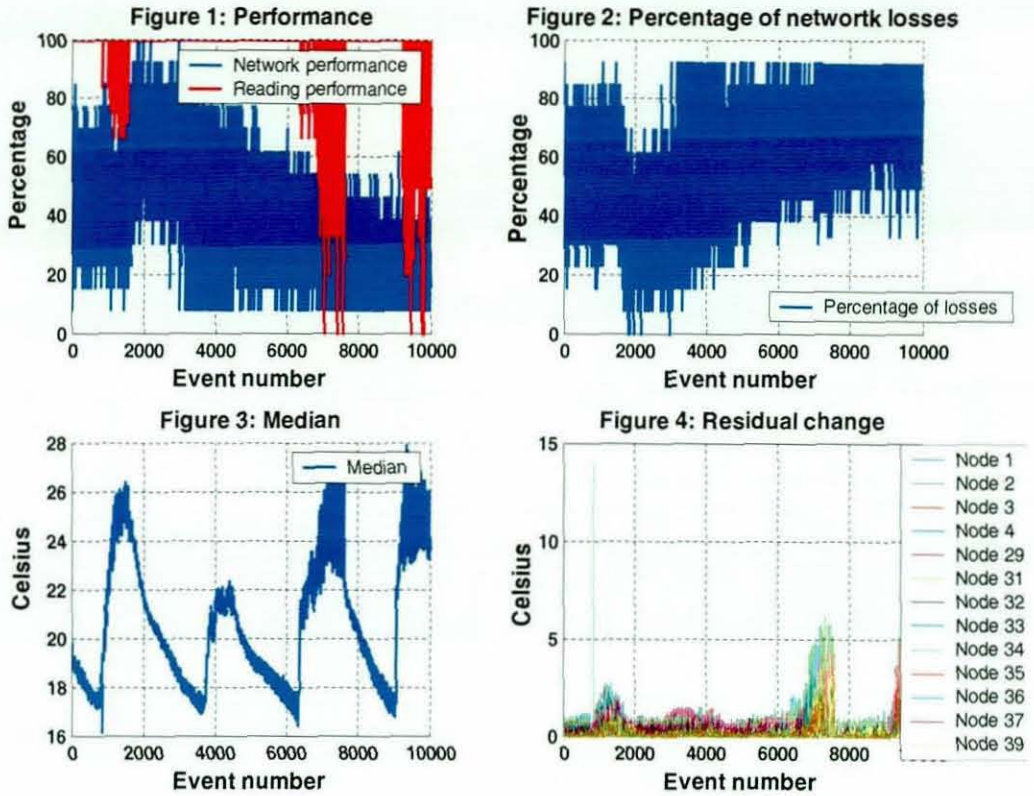

Figure 7-33. The Algorithm Operation without Power Saving

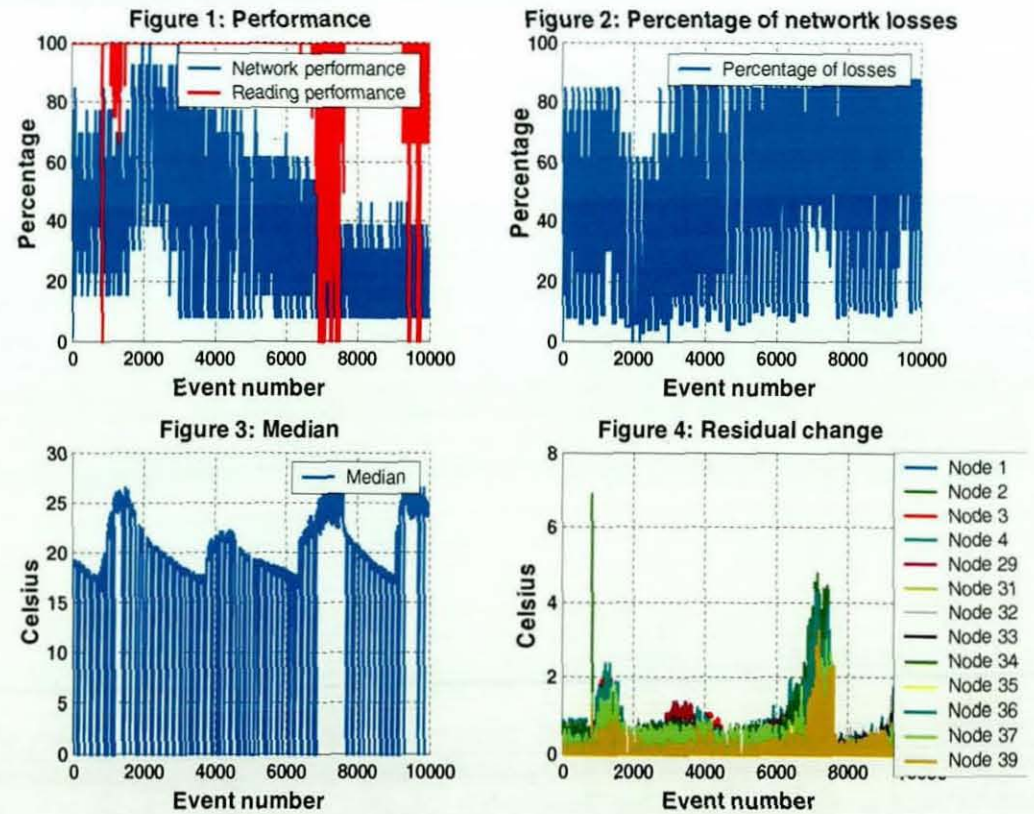

Figure 7-34. The Algorithm Operation with Power Saving

7.6 Summary

The results of the simulation experiments showed a high level of accuracy in the algorithm's detection of both permanent and temporary deviations that had a direct impact on the network's functionality and degraded the network's performance. These experiments showed that the proposed algorithm was able to detect, on average, from 64% to 97% of the faults detected by statistical methods. The other undetected faults arose as a result of the occurrence of more than one deviation concurrently; i.e. considered as a phenomenon change due to the correlation between them.

Also, these simulations show that, if the faulty nodes are not removed and their number exceeds 50%, the algorithm will make wrong detections due to faulty readings causing a deviation from the median calculation. This is can be clearly seen when the readings of healthy neighbour nodes are lost and most of the received readings are deviated ones. On the other hand, removing them causes a reduction in the network's connectivity and increases the impact of any reading deviation, depending on the degree of change and the number of neighbours.

Finally, the chapter shows that there was a 15% to a 38% reduction in the algorithm's power consumption if a power-saving method of detection was implemented. However, this came with a higher degree of algorithm detection false which had an impact on the accuracy of neighbour readings; it also took a longer time for faults to be detected.

Chapter 8 Empirical Experiments

8.1 Introduction

This chapter describes the practical implementation of the proposed algorithm on the Berkeley (Crossbow) Mica2 sensor Motes testbed [116], programmed by *nesC* on the *TinyOS* operation system. The main target of this implementation is to test the functionality of the algorithm under real sensor network scenarios that cannot be simulated as a result of their unpredictable behaviour due to losses, the different effects of the wireless channel and dynamic topologies, node constraints resources, and environmental changes. The impact of these different factors was compared with the outcomes of the simulation experiments recorded in Chapter 7 to check the consistency of the results.

This chapter starts by describing the platform used in the experiments, the *nesC* program, and the *TinyOS* communication stack. Then it discusses the methods of implementing the algorithm and the necessary modifications for different protocols and communication stacks of the multi-hop *'Surge'* continuous reporting and event-driven application. This is followed by a discussion of some of the practical difficulties faced while implementing the algorithm on the testbed. Finally, the chapter considers the outcomes from experiments using one-hop and multi-hop networks before offering conclusions.

8.2 Platform Hardware

Crossbow [116] is considered to be one of the leading companies in the field of commercial sensor network solutions (such as MicroStrain [117], Ember [118], and Millennial Net [119]). The company offers various *WSN* products for different applications with three main building blocks: i.e. sensors, processor/Radio boards called Motes, and gateway/network interfaces.

A Mote consists of a power unit, a transceiver unit, a microprocessor, Analog Digital Converter (*ADC*) and a 51-pin expansion connector that supports

analog inputs, digital Input/ Output (*I/O*), Synchronous Serial Port (*SPI*), and a Universal Asynchronous Receiver/Transmitter (*UART*), as shown in Figure 8-1. These Motes have different models depending on their radio characteristics and the interfaces they use. These are listed in Table D.1 in Appendix F.

The testbed used for the experiments consisted of 15 Mica2 nodes along with a 13 MTS300CA sensor board and 2 MIB510 gateways that were used to program Motes and to collect data from the sink.

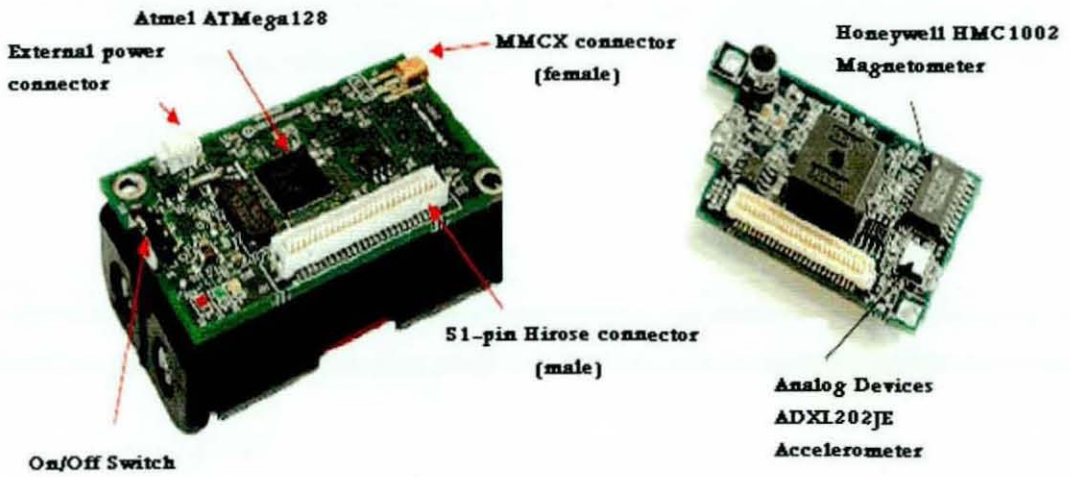

Figure 8-1. Mote Node and its Sensor

8.3 Platform Operation System (*TinyOS*)

UC Berkeley designed an open source, event-driven execution module system called *TinyOS* [17],[18],[120],[121] to work with wireless embedded sensor networks. It was designed such that its flexibility increases at the design stage and its usage of resources reduces at the operational stage. This is achieved by providing a framework for constraining modular resources and event-driven concurrency modules that have the ability to manage the hardware capabilities of *WSN* nodes. This framework reuses sets of components and connects them with each other by wiring specification and

configuration modules. These reused components are gathered from the TinyOS library (i.e. they are written in the nesC language) and includes network protocols, distributed services, sensor drivers, and data acquisition tools. Moreover, this framework uses two different types of event-based concurrency module tasks that can post the task in a queue and postpone its execution until the scheduler makes a request, or a hardware event handles the response. These event-based concurrency modules can be used to overcome speed issues between logical and physical worlds.

8.4 *TinyOS* Communication Stacks

The basic method of communication provided by the *TinyOS* in *WSNs* is by Active Messages that are provided by two *TinyOS* library modules [122]. These are the 'Generic_Comm' component, that builds header components and control packets that specify the destination of the packet; and 'AMStandard', which indicates when a transmission has been successful. These messages are based on networking abstractions where each includes an identifier that specifies the action to be carried out upon reception.

An Active Message consists of payload data and a header, as shown in Figure 8-2. Its size varies from one application to another but in general it has a header size of 5 bytes (i.e. 2 for the message destination address, 1 for message type, 1 for group identification (ID) that represents all nodes working in the same application, and 1 for message length), and a payload size of 29 bytes (i.e. 2 bytes for the source address, 2 for counter, 2 for ADC channel and ADC readings, and 2 bytes for the Cyclic Redundancy Check (CRC) that is used to ensure the reliability of the transmitted data).

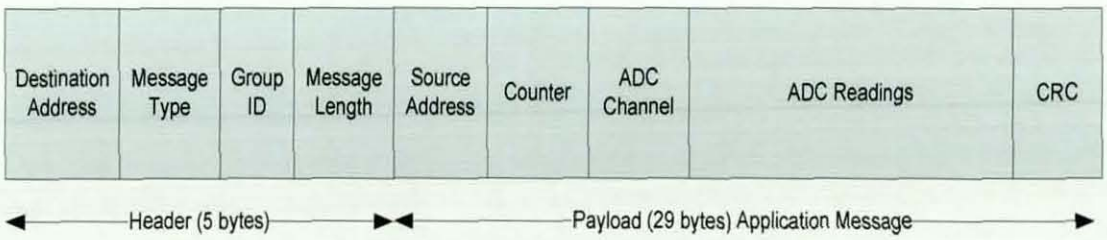

Figure 8-2. Active Message Structure

8.5 Warning Packet Design

Since the algorithm is designed to be used for large-scale Wireless Sensor Networks offering either individual node or cluster-head node reports, the warning packet is designed as a multi-hop packet and follows its structure. This is because most large-scale Wireless Sensor Networks (*WSNs*) use a multi-hop routing protocol[13].

Also, there is a possibility that the proposed algorithm can be used with single-hop applications (these deliver all sensed data to a base station), but it is more accurate and uses fewer network resources if external analysis software, such as MOTE VIEW [89], History Viewer [88] or another prediction-based tool, is used at the sink.

8.5.1 Surge Application

The *'Surge'* application [18], [120] is a multi-hop source-driven application that periodically reports its readings (i.e. temperature/light measurements) over a mesh network topology to a base station. Sensor nodes in this application have the ability to respond to commands that are broadcast from a base station. This allows the base station to control the sleeping period of nodes and vary their reporting rate.

This application was chosen for use with the algorithm because it does not need a special node capability, such as location awareness/*GPS*, as it does not make any assumptions concerning node distribution. Moreover, it consists of homogenous nodes that are of equal importance, and operates correctly even if nodes are not synchronised. The main challenge of these types of application is the construction of a table of neighbour measurements and methods of arranging them; this depends on the time their packet is received, as discussed in Chapter 4.

---

[13] This is because combining multi-hop features with battery management at the sensor node helps with the network's self-configuration; it also helps self-heal nodes, makes the network scalable, extends the network's lifetime, and reduces the complexity of node deployment [104].

The '*Surge*' application communication stack is divided into three. Its main packet consists of four different fields; Figure 8.3 (i.e. two synchronous headers, packet type, and payload data). The payload field includes *TinyOS*, multi-hop, and application (such as Surge messages that vary the size of the send packet). The hexadecimal packet nodes are exchanged at certain times and an instance of this can be seen in Figure 8-4.

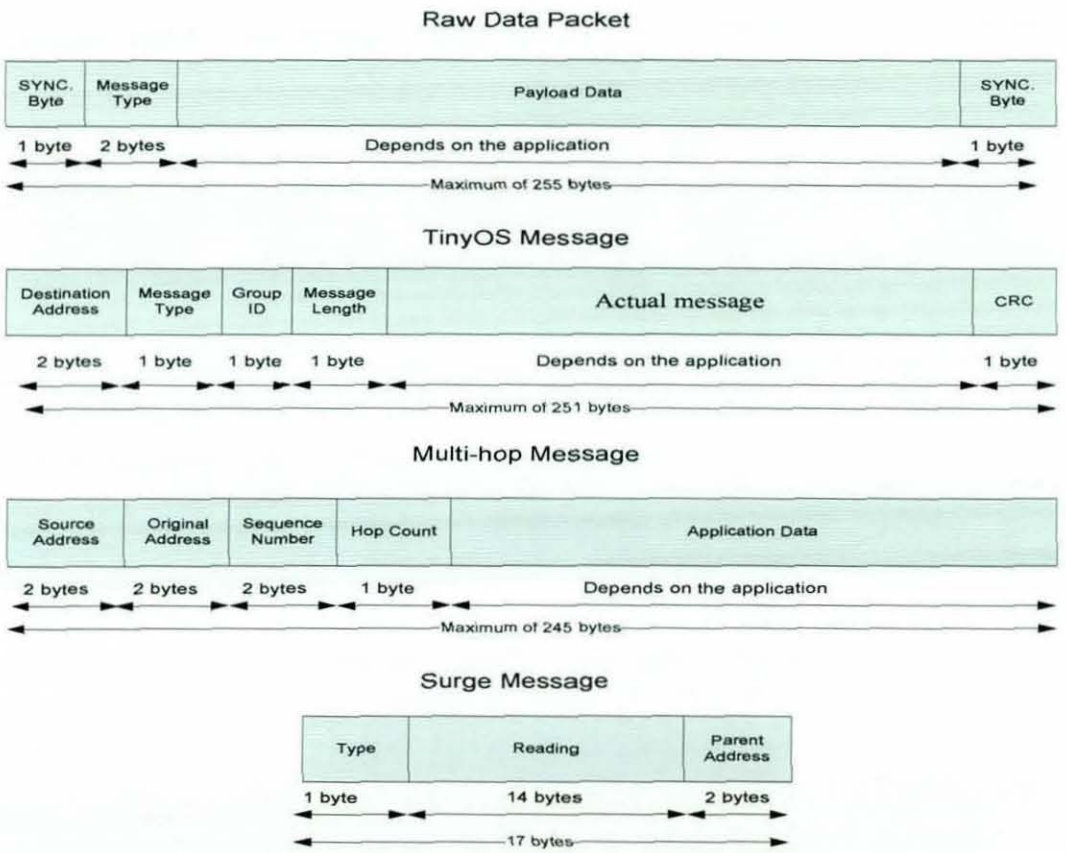

Figure 8-3. *TinyOS* Packet Structure

FF FF FA 7D 1C 03 00 03 00 2B 00 02 01 00 06 01 00 FF 04 00 FF 05 00 FF 02 00 F9 06 00 F8 00 00 CC
00 00 11 7D 0C 01 00 01 00 2E 00 01 00 A3 03 00 00
00 00 11 7D 0C 02 00 02 00 2D 00 01 00 B8 03 00 00
FF FF 11 7D 0C 06 00 06 00 28 00 FF 00 9C 03 FF FF
FF FF 11 7D 0C 05 00 05 00 29 00 FF 00 AB 03 FF FF
FF FF 11 7D 0C 04 00 04 00 2A 00 FF 00 AD 03 FF FF
FF FF FA 7D 1C 04 00 04 00 2B 00 02 02 00 06 02 00 FF 05 00 FF 06 00 FF 01 00 F9 03 00 F3 00 00 CC

Multi-hop Packet Type
Surge Packet Type

Figure 8-4. Different *TinyOS* Packet Size

In order to send a warning packet, a new packet message is added to the three original '*Surge*' packet types. This packet carries the algorithm's

detection parameters with a total length of 8 bytes, as shown in Figure 8-5. The first field of this warning packet is assigned for the total number of nodes, including the monitor node itself. The second and the third fields indicate the detected number of suspected dead and deviated nodes respectively. This is then followed by the address of suspected node and the type of fault. This fault detection is divided into 11 types, as shown in Table 8-1. The last field in the warning packet contains the number of times the suspected node has been detected as a fault. This warning packet is designed so that it follows the *TinyOS* multi-hop addressing and exchanging stack, as can be seen in Figure 8-6.

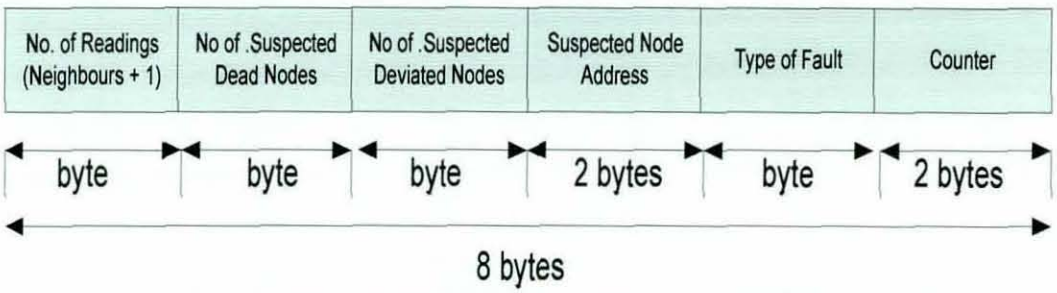

Figure 8-5. Warning Message Packet Combinations

Fault Type	Code
TOPOLOGY_UNSTABLE	0
FAULT_TYPE_DEVIATION	1
FAULT_TYPE_COMMUNICATION	2
FAULT_TYPE_COVERAGE	3
FAULT_TYPE_ENERGY_CONSUMPTION	4
NO_EVEDENCE_OF_FAULT	5
FAULT_MESSAGE_STOP	6
FAULT_TYPE_DEID	7
FAULT_CLEAR	8
NEIGHBORHOOD_MULFUNCTION	9
PROTOCOL_EFFECT	10

Table 8-1. Code of Fault Type in the Warning Message

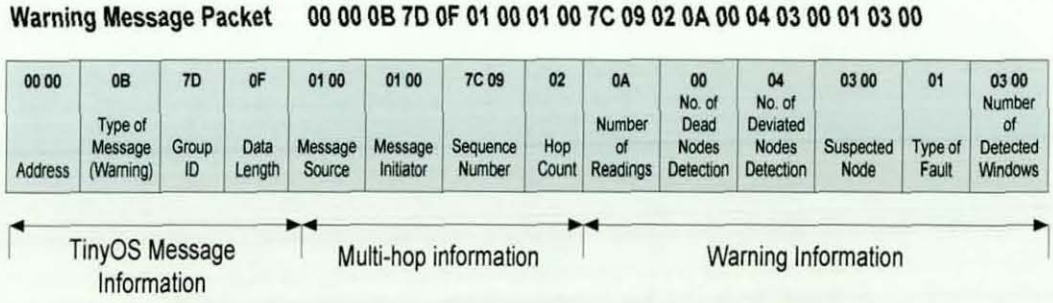

Figure 8-6. Warning Message: Different Fields

8.6 Programming at the TinyOS Multi-hop

As discussed in Chapter 4, the proposed algorithm can be implemented either at both application and communication levels or at a communication level where measurements are pushed down to it from the application layer.

At a communication level, implementation is carried out directly on a *TinyOs* Multi-hop application [15]. This application consists of the 'MultiHopEngineM' which is used to provide the overall packet movement logic for multi-hop functionality while 'MultiHopLEPSM' is used to provide the link estimation and parent selection mechanisms, as shown in Figure 8-7.

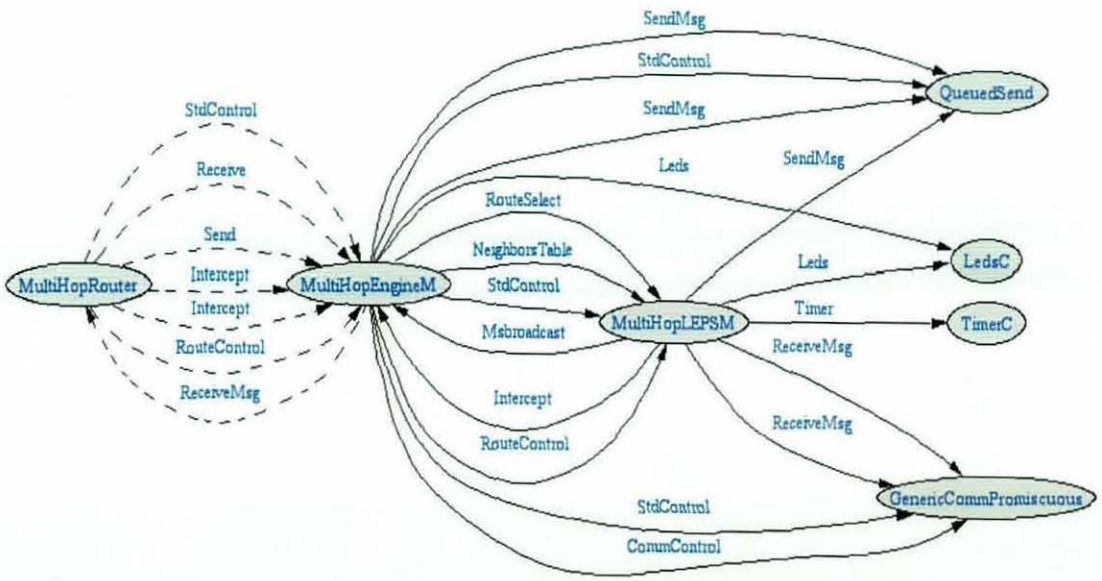

Figure 8-7. Multi-hop Component Interfaces

These two modules were modified by adding the functions of the proposed algorithm, as shown in Figure 8-8. This modification was made in two of the modules of the multi-hop protocol components to reduce the total memory usage, to reduce the data exchange between different modules, to reuse the existing memory storage and to reduce the time required for data processing. The algorithm was programmed so that there would be links between its different modules in the two multi-hop components (i.e. 'Msbroadcastand' and 'NeighborsTable' interfaces components) to exchange information between

the application's components in order to reduce the usage of the *RAM* resources especially by utilising the local stored application parameters.

The proposed algorithm's implementation in method 1 added a total of 337 lines to the *'Surge'* application's source code with 13 functions, four of them as a table construction. This is shown in Table 8-2.

Compiling the software shows that there was an additional 1650 bytes in *RAM* (i.e. around 82.3% more due to the use of *RAM* slots in the algorithm functions and analysis) and 6802 bytes in *ROM* (i.e. around 38.2% more), as shown in Table 8-3[14]. This memory size can be reduced in synchronised, aggregation/fuse applications due to using synchronising alignment in neighbour packets which the Surge application does not have.

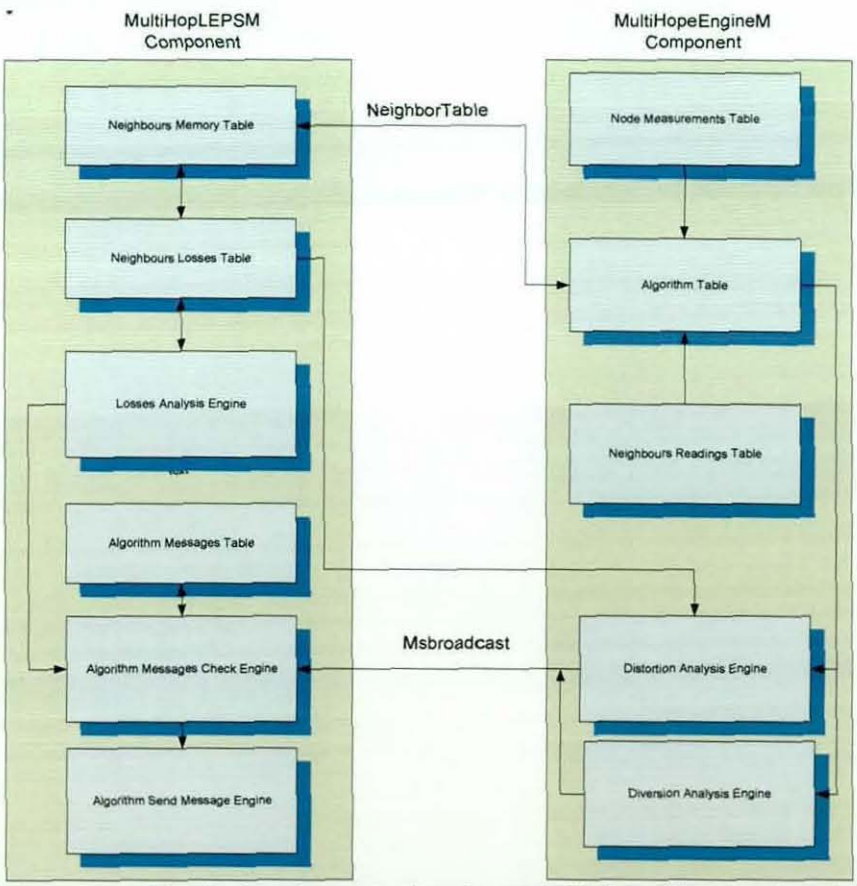

Figure 8-8. Algorithm's Implementation in a Multi-hop Protocol and the Link between its Different Modules

[14] These values were taken from the screen output after compiling the source code (i.e. memory footprint).

The second implementation method used an advantage noted by Sankarasubramaniam *et al.* in [114]. They argued that expanding the packet size to a certain percentage would result in only a very small increase in the energy consumption due to the negligible energy consumption of the node transceiver in that short period. Method 2 of the proposed algorithm implementation was designed to push down application measurements from the application layer to the communication layer and insert these into the routing update packets broadcast in the neighbourhood, as shown in Figure 8-9.

MultiHopEngine			MultiHopLEPSM		
Function	**Number**	**Lines**	**Function**	**Number**	**lines**
Reading Table	3	61	**Loss Table**	1	14
Median & dynamic threshold	1	51	**Loss analysis**	1	45
Distortion analysis	1	5	**Fault track**	1	6
General	------	11	**Send messages**	1	41
Diversion analysis	-----	51	**Faults analysis and reply**	1	21
			Stop messages		7
			Clear faults	-----	6
			General	3	18
Total lines			337		
Total Functions			13		

Table 8-2. Functions and Number of Lines Added to the Algorithm Functions in a Surge Application (Method 1)

The implementation in Method 2 was achieved by modifying the application code, as shown in Figure 8-10. This modification reduces the size and the complexity of the algorithm, as shown in Table 8-3, where *RAM* utilisation was reduced by 4%, *ROM* utilisation reduced by 14%, the total number of source code lines reduced by 8% and the number of functions and tasks reduced by 114%. Moreover, this modification made the algorithm independent of the

application's reporting rate and dependent on the update rate of the routing protocol.

	Without the Algorithm	Method 1		Method 2	
		With the Algorithm	Addition Percentage	With the Algorithm	Addition Percentage
ROM (bytes)	17760	24552	38.2%	23938	34.8%
RAM (bytes)	2004	3654	82.3%	3368	68%
Total number of lines	2860	3197	11. 8%	2977	4%
Functions & tasks	7	21	200%	13	86%
Modules	31	34	8.8%	34	8.8%

Table 8-3. Memory Used Before and After Adding the Algorithm to the Surge Application

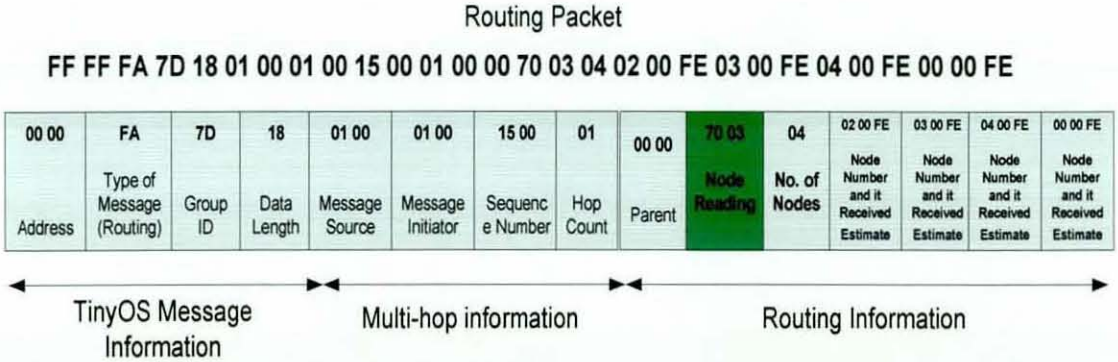

Figure 8-9. Construction of the Multi-hop Routing Packet of the Proposed Algorithm Using the Second Method of Implementation

Also, the proposed algorithm can be programmed by creating new specified components instead of integrating the algorithm's function in the existing *TinyOS* components as discusseed above. The advantage of this approach is that it has a special component for the proposed algorithm that can be reused when required without using a multi-hop protocol and the *'Surge'* application. The disadvantage of this approach is that it uses additional resources such as memory and processing that were saved when the algorithm used the local parameters in *'Multi-hop'* and *'SurgeM'* components.

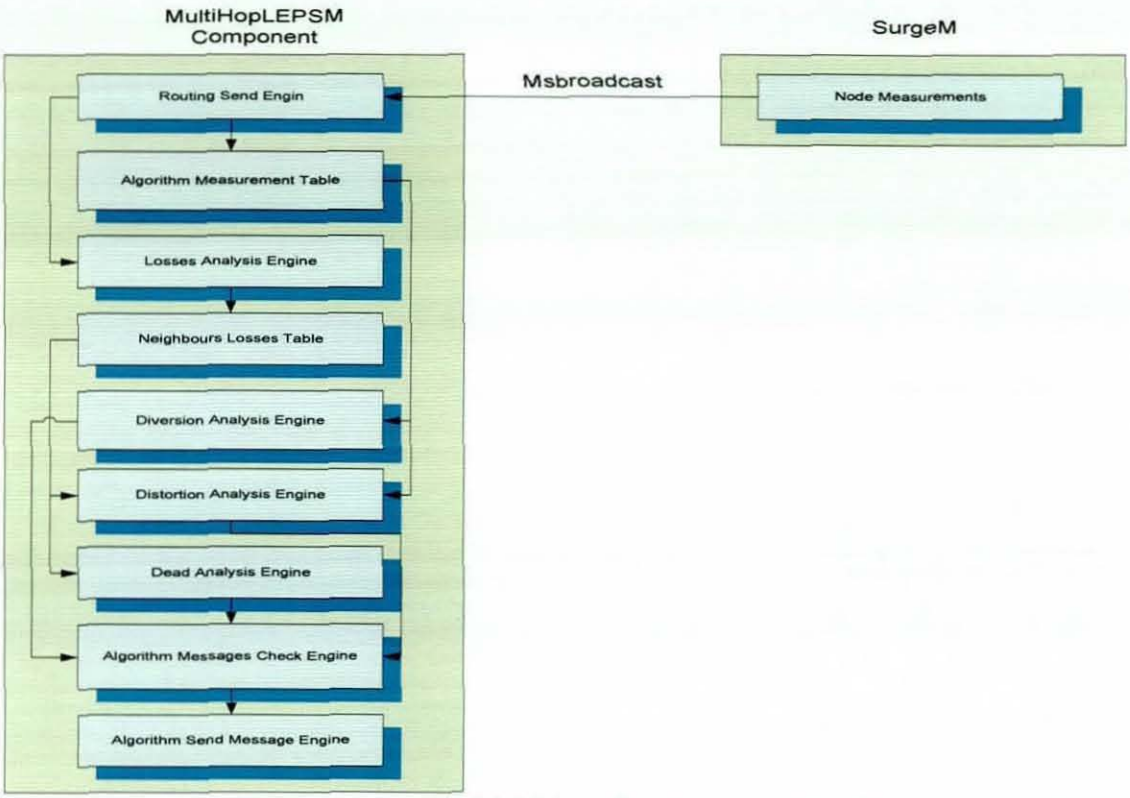

Figure 8-10. Implementation on a Multi-hop Protocol and the Link between its Different Modules

8.7 Difficulties in Practical Implementation

Many difficulties were faced while implementing the proposed algorithm practically on the Mote2 platform. These included source code debug, parameter control, resource usage and many others.

8.7.1 Source Code Debug

One of the main problems faced while implementing the algorithm was its debugging. This is because the event-driven approach that *TinyOS* uses makes the separately used threads in the source code process all tasks in disjoint stages. This means that these threads do not represent the control flow for the processing of a particular task. Also, the state of these tasks is not stored in local variables in the stack in a threaded system but is handled in the task itself [17].

In general, debugging can be carried out by analysing the output experiment's trace file which is collected from the snoop/sink/node by a second program

such as *MATLAB*. The main problem with this method is that the collected trace file is not necessarily the same for all the nodes especially at high loss rates, and this method will not give the individual node variables that are available.

To overcome this problem, the nodes' local source code can be modified such that it sends the necessary monitor node variables when they occur to the node serial port or via a communication stack to the sink. This method has two main problems: first, there is a need for the node to be connected to the serial port of a *PC* in order to collect these variables (or send the results to the node *EEPROM*); second, this method will add further resource usage, such as processing time and energy, to the actual node function usage.

Finally, *TOSSIM* and *PowerTOSSIM* simulations [47], [115] can be used for debugging by sending the node variables to the simulation output trace file without adding additional resource usage to the main code. The problem with this method is that it is not guaranteed that the tested code will work in practical terms especially as these simulations work with faster processors and PCs with a larger memory size than the real sensor platform.

All the above-mentioned methods were used to debug the proposed algorithm in order to avoid the drawbacks of each one.

8.7.2 Code Complexity and Usage of Resources

The other critical problem faced while implementing the algorithm was code complexity and its memory needs. Although there are several solutions for detecting and tracking malfunctions in sensor nodes, such as relating losses to node readings and comparing them to the other neighbours in the neighbourhood, their source code is complex and requires a large memory space that the existing *WSN* node platform cannot handle. This complexity needs to be reduced to fit the capacity of the node memory and the processor. Although some of algorithm works efficiently in a *TOSSIM* simulation, its practical implementation on the testbed was unsuccessful.

From the practical implementation of the algorithm (on a Mica2 platform), it was found that, as the source code becomes more complex, network losses increase and there will be a collapse in the network's functionality after a certain time or the application will not work totally when deploying nodes.

8.7.2.1 Time Alignment

In *TinyOs*, components are built one on top of the other; this provides a set of functions which are interdependent and so time alignments between these different components tasks are essential [123]. Any unmatched alignment causes a sudden crash of the application, for example employing a simple loop for time delay may cause the application to crash and stop the operation of nodes if the loop delay period is not matched with other application tasks during that period. This reduces the flexibility of *TinyOS* and prolongs source code and memory usage. For example, when there is more than one fault detection, warning messages cannot be sent instantaneously after the detection and memory space is then needed to store these up to the time they are sent. Moreover, concurrency needs to be used to check when it is time to process the task and send messages.

8.8 Special Issues Faced in the Practical Experiments

The experiments showed that there is a relation between the detection of node malfunctions, network functionality detection, detection response time, detection threshold, and the size of the monitoring window. There has to be a balance between these parameters in order to achieve high performance in detection. For example, an increase in the algorithm's threshold value leads the proposed algorithm to detect node malfunctions and not degradation in the neighbourhood nodes' coverage of the measured phenomenon, as discussed in [25]. Moreover, as the algorithm becomes more complex, its analysis becomes more accurate, but at the cost of higher levels of resource usage; this reduces the network's lifetime. This complexity also causes an increase in packet losses due to process failure, as discussed in [13]. Although the level of confidence of the algorithm's detection can be increased

by increasing the size of the monitoring windows, it takes a longer time for the proposed algorithm to detect the degrade in the network's functionality which increases the impact of node malfunctions on the network's performance.

In order to set these parameters practically, many challenges must be taken into consideration. The following section summarises these challenges.

8.8.1 Threshold Setting

There are two methods for threshold setting in the proposed algorithm: static and dynamic. The static method depends on the threshold setting before the nodes' deployment. With this method there is no need for an on-demand setting and its value depends on the required node coverage, network connectivity, topology dynamics, and the application's characteristics, as discussed in Chapters 2 and 4. However, this comes with the difficulty of setting the correct threshold value especially with a low density of node deployment and high levels of change in the characteristics of the monitored phenomenon, such as light intensity measurements which cause the release of a large number of warning messages, as shown in Figure 8-11. This figure presents the results of the experiment conducted with 10 sensor nodes measuring light intensity, distributed over an area of a 6x6 square metre and each with a 5 dBm transceiver output power. Due to the low number of nodes available in the testbed, the experiment was repeated, changing the threshold values until the warning messages stopped. (This was increased to a threshold value of 30% of the calculated neighbourhood median).

This threshold method can be used at the deployment stage when there is a need to check that the targeted deployment goals are satisfactory. It can also be used to guide the administrator to those areas that need more nodes and to test the degree of dynamic of the topology, the connectivity of the deployed nodes due to the surrounding environment, and the degree of node coverage.

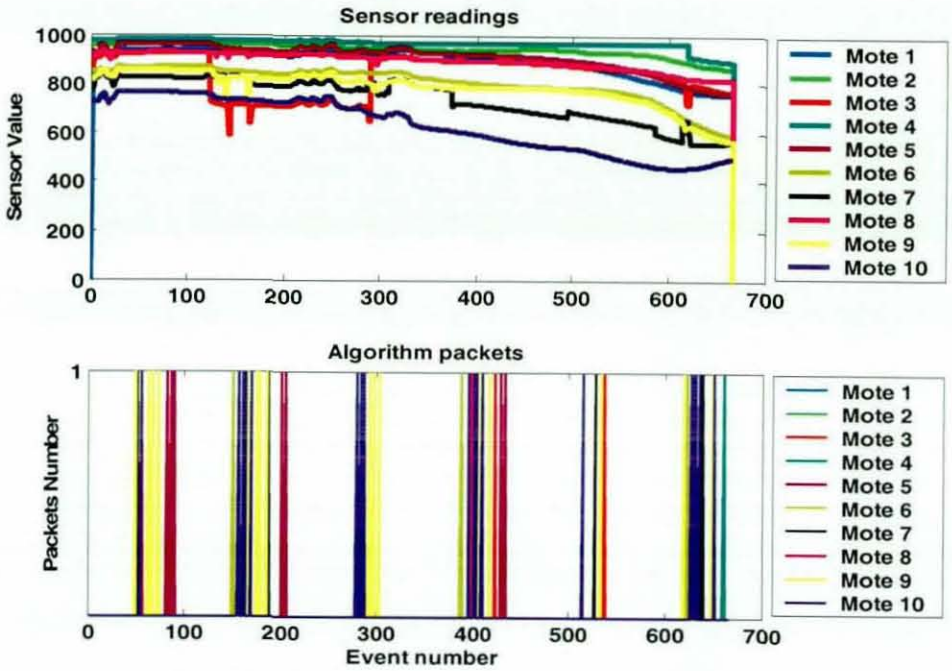

Figure 8-11. Experimental Results Using a Static Threshold

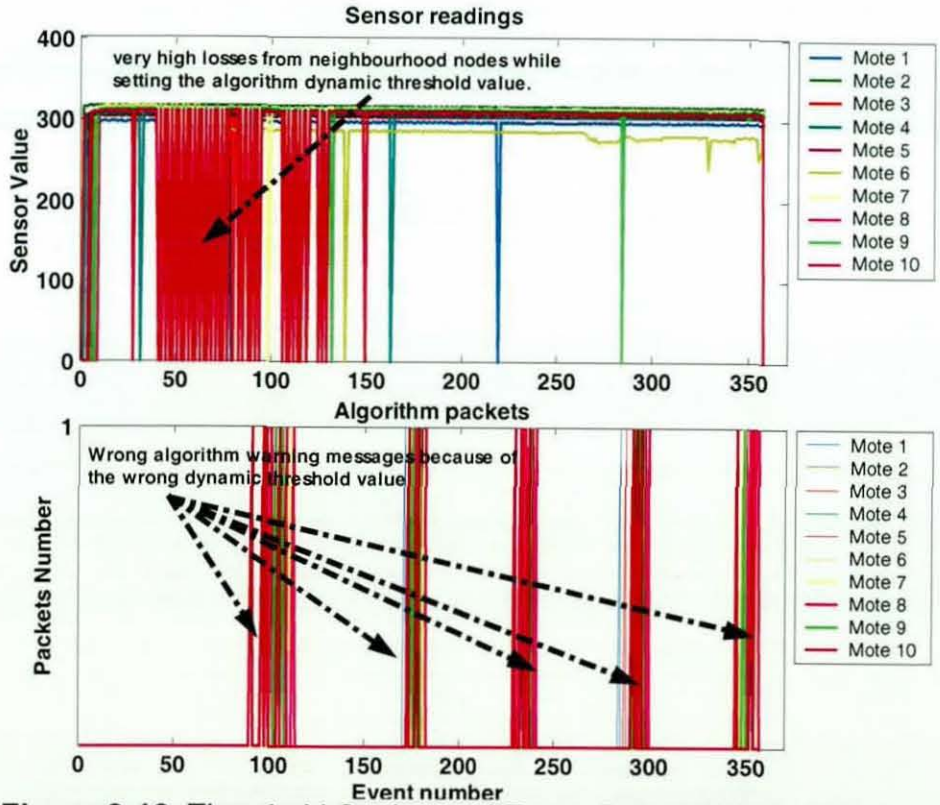

Figure 8-12. Threshold Setting and Errors Due to Different Protocols Handshaking at an Initial Stage

The second method uses a dynamic threshold that depends on the neighbour measurements received. This approach to set the threshold depends on the

circumstances in the field regarding deployment and takes the threshold as a reference. Any change from this reference is detected as a change in the node coverage or as a node malfunction. The main problem with this method is the impact of neighbours' packet losses at the time when the threshold is set, as shown in Figure 8-12. This figure shows a high level of losses at setting time that causes neighbour nodes to have different threshold values; this makes the algorithm's analysis inconsistent as it then releases wrong warning packets.

This problem can be solved by added a waiting time for the handshaking period. This solution improves the threshold setting and reduces the error. However, with applications with a low reporting rate, this waiting time causes the period required to monitor neighbour nodes to be longer, depending on the application's reporting rate and threshold closeness required between neighbour nodes' readings.

8.8.2 Losses and Internal Parameters

Losses occur in sensor networks because of the common wireless medium or due to the use of the event-driven *TinyOS* where its event-driven nature can cause process failure[15] and may alter the parameter values of internal functions, as discussed in [13]. The alterations in the parameter values arise as a result of the lack of a dedicated I/O controller in *TinyOS* and the unavailability of memory protection; this makes it easy to become corrupted, as discussed in [13], [17]. This feature has been detected in some of the empirical experiments. For example, Figure G-1 in Appendix G shows the hexadecimal output of experimental messages that indicate the detection, by Node 2, of a deviated Node 3 when the deviated node was actually Node 8; this was detected by other nodes in the neighbourhood.

The empirical experiments that were conducted showed that losses and the altered parameter values increased as the number of nodes sharing the same medium increased, as the code became more complex, and as the reporting

[15] Process failure occurs when the sensing operation is interrupted by other network tasks [13].

rate increased. The high losses caused the message that was sent from some nodes to be repeated (as shown in Figure 8-13). In some experiments, the repetition of the sent packet continued for a long time and up to the initialisation of the sending node. This effect was reduced by reconstructing the warning packets in multi-hop format, using short tasks with an *'atomic'* function to ensure the processor was forced to finish the task before going on to another. The effect was also reduced by storing the content of the send packets in the memory until the transceiver was free.

```
....
00 00 11 7D 0C 09 00 08 00 28 01 01 00 BD 02 09 00
FF FF 0B 7D 06 07 00 05 00 07 03
FF FF 0B 7D 06 08 00 05 00 06 07
FF FF 0B 7D 06 0A 00 05 00 06 07
FF FF 0B 7D 06 09 00 05 00 06 07
FF FF FA 7D 1C 07 00 07 00 CC 00 02 04 00 06 08 00 FF 04 00 FE 09 00 FE 05 00 FE 01 00 FD 03 00 FD
....
```

Figure 8-13. Example of Message Repetition

Moreover, these experiments showed losses in the proposed algorithm warning messages sent to the sink and occurred either with multi-hop routing or at the node itself. This was solved by sending more than one warning message from different neighbours that carried the same message, thus increasing the probability of its arrival at the sink (also it will increase power consumption).

In order to check the impact of extra warning packets, randomly distributed nodes, with different densities range between 500 to 1400 nodes, were simulated in an area of 1000X1000 square metres using multi-hop reporting to the sink. Each node had a 50 metre sensing and transmission range. These experiments used MATLAB code for simulation and the power consumption model shown in Appendix A. A combination of distributed election leader routing and distance-vector routing, as described in [23], were also used. Table 8-4 shows the overhead of the average path length, for a warning packet, and the average number of nodes receiving the broadcast warning throughout the path; with different networks densities. As can be seen from the table, energy consumption increases as the node density increases, as well as the number of nodes receiving the broadcast warning throughout the

path. This increase is linear for both average path length and average number of nodes receiving the broadcast warning throughout the path, but with different speeds.

Deployment	Node density		Send Warning packets		Received Warning packets		Total Energy mJ	
	AVR	STD	AVR	STD	AVR	STD	AVR	STD
500	4	2	6	5	27	54	1.7	1.5
600	5	3	12	8	63	169	2.5	3.9
700	5	3	17	8	108	341	3.9	6.9
800	6	3	22	7	129	419	4.8	8.2
900	6	3	22	9	149	510	5.2	10
1000	7	3	19	8	163	603	5.1	11.5
1100	8	3	24	7	189	780	6.1	14.5
1200	9	3	20	8	197	767	5.8	14.4
1300	10	3	20	7	203	795	5.9	14.8
1400	10	3	20	7	211	846	6	15.7

Table 8-4. Overhead of the average path length, for a warning packet, and the average number of nodes receiving the broadcast warning throughout the path; with different networks densities and a combination of distributed election leader routing and distance-vector routing as described in [23][16]

8.8.3 Window Setting

To detect the health of the network, three windows were created, as shown in Figure 8-14. The small window checks how many deviations are detected within that period and is more sensitive to temporary, short-term deviations, while a big window gathers detections from the small windows so that it increases confidence in detection, as discussed in Chapter 4. A third window was designed to monitor the exchange messages so that the number of messages which were exchanged were reduced by stopping or releasing them.

[16] Please note that the results are representative for one run of the experiment. AVR is the average and STD is the standard deviation.

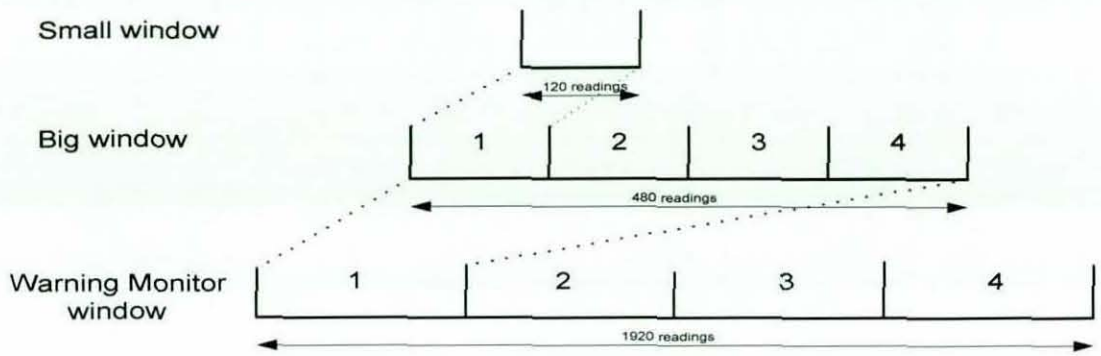

Figure 8-14. Algorithm's large and Small Window Constructions

The experiments showed that a small window is not suitable for sending fault warnings due to the frequent node malfunctions in the *WSN*. If such windows are used there will be a large number of false algorithm warnings and a lot of '*No_ Fault_Evidence*' messages will be released. Moreover, increasing monitor window size causes fewer false algorithm warning messages but, along with this, there may be no detection, as happened during one of the experiments when all the nodes were dead within the period of time when detection was monitored, without any of the dead nodes being reported. To solve this problem particularly with applications using low reporting rates, data validation tests, along with a small window size, could be used to increase algorithm's detection confidence, as discussed in Chapter 4.

8.8.4 Warning Message Exchange

Several experiments were conducted to check the efficiency of warning message exchange. Several problems were faced during these experiments:

- The set and reset of warning messages due to the effect of a highly dynamic network. This is solved by adding a counter that sends a warning message to the sink stating that there are fluctuations in the monitoring of suspected nodes; it then stops sending it. Moreover, if a node sends a *'NO_FAULT_EVEDINCE '* message to a reporting node, it will stop this report because of the no evidence reply from the monitoring node neigbour.

- If a node started at a different time or initialised during the process, it will have a different threshold value and a different monitoring window time than others. Also, it may release more warning or *'No_Fault_Evidence'* messages. This can be solved by synchronizing the monitoring time between neighbours at the start, either by a message sent from the sink or start algorithm analysis after a certain waiting time.

8.9 Tests on the Algorithm

Several experiments were conducted indoors at the High Speed Network Research Group *Lab* in Loughborough University to test the proposed algorithm's functionality in a real sensor network. These experiments were conducted in the presence of other devices that are able to interfere with the sensor transmission. The node antennae were bent at the top to reduce the range. These offer experiments in a dynamic topology and in circumstances of high packet losses. These experiments were organised in a one-hop configuration to test the functionality of the algorithm under different packet losses, and in a multi-hop configuration to test the functionality of the algorithm under a dynamic topology.

8.9.1 One-Hop Experiments

These experiments were conducted with a number of nodes varying between 2 and 13, and at different distances from the sink and from each other. Nodes were arranged in straight lines, in circular and in random formations (i.e. to have different loss percentages). They were programmed at the 0x09 power range (i.e. -10 dBm). A snooping node was added to the network to monitor all the packets that were exchanged, including the network routing packets, it was programmed at 5 dBm. Sensors in these experiments were measuring light intensity.

The aim of the experiments was to test the algorithm detection under different scenarios at the node level. Three metrics were chosen in order to analyse the results. The first was the average percentage of correctly detected faults,

which measures the ratio of the number of deviated faulty sensors that are diagnosed as faulty nodes, compared with the total number of faulty sensors in the network. This metric computes the ability of the algorithm to detect faults. The second metric chosen was the average positive false detection. This measures the ratio of the number of healthy nodes diagnosed as faulty, to the total number of nodes diagnosed by the algorithm as faulty. This metric computes the errors detected by the algorithm. The third metric was the average percentage of negative false detections; i.e. the ratio of faulty deviated nodes that were not diagnosed as faulty, as opposed to the total number of faults in the neighbourhood. This metric records the algorithm errors in not detecting faults.

All the metrics discussed above show the impact of an increasing number of faults on the algorithm's detection, together with the impact of changing the algorithm's detection threshold on deviation detection and warnings released by the algorithm. The faults in these experiments were inserted into nodes by covering the sensor with paper. Dead nodes were created by switching off the node.

8.9.1.1 Algorithm Fault Detection

Figure 8-15 demonstrates changes in node light intensity measurements and in warning packets in an experiment that arranged 4 nodes in a straight line at a distance of 15 cm from each other. The figure shows the detection accuracy when the algorithm sent warning messages for the deviated Node 1 and the dead message for Node 4.

8.9.1.2 Detection Performance for the Neighbourhood

As mentioned previously, the algorithm's warning packet contains in its fields the number of deviated nodes and the total number of nodes in the neighbourhood. This gives the expected network performance in terms of the accuracy of the collected readings and the nodes' connectivity. Figure 8-16 plots the measurements of the neighbourhood nodes, the detection of deviated nodes, and the degradation in the accuracy of the collected data due

to these deviations. This was obtained from an experiment using 10 nodes measuring light intensity arranged in circle. The figure shows that the accuracy of the collected data was degraded after inserting faults. This degradation depends on the change in the number of healthy nodes in that neighbourhood. This indication of degradation continues up to a level of 50% deviated neighbours then starts to fluctuate. This fluctuation depends on the percentage of deviation from norm of each node, the number of deviated nodes, and losses received among neighbours' packets.

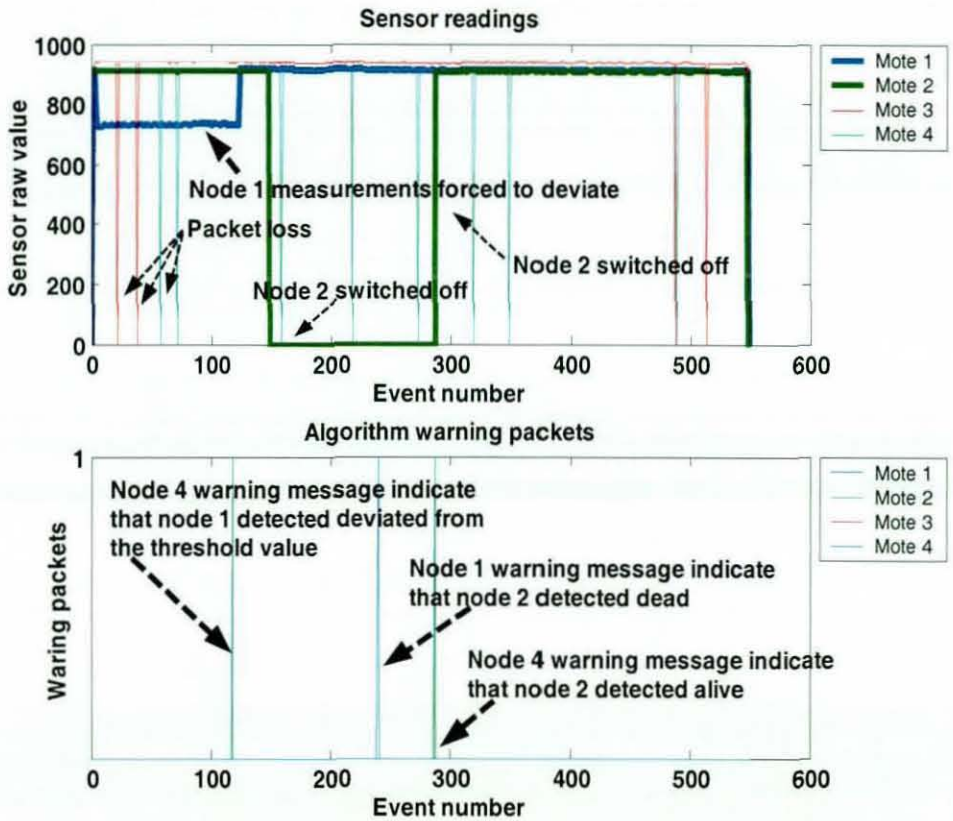

Figure 8-15. Algorithm's Deductibility Test with 4 Nodes

Figure 8-17 plots the contents of the warning packets received from the 10 nodes and shows that all the nodes in the neighbourhood detected the same degrade of network performance up to a level of 50% deviated nodes. When the number of deviated neighbours increased to over 50%, the nodes were divided into two groups in terms of the value of performance measurements (i.e. nodes 10,9,8,5 and nodes 7, 6, 2, 4, 3, 1) such that each group of nodes were physically close to each other. This showed that the algorithm detection depends on the number of readings at any time interval, the degree of

deviation and the losses of healthy readings at certain time intervals. These results agree with the findings of the simulation in Chapter 7.

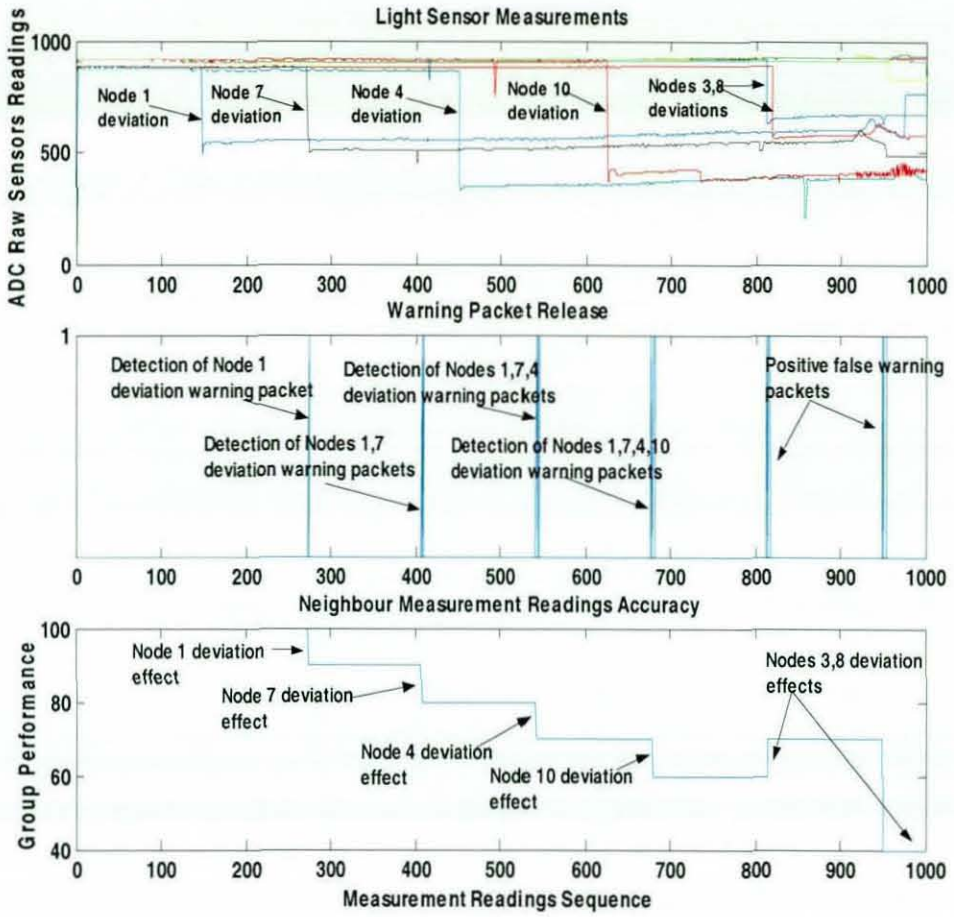

Figure 8-16. Network Performance Detection

8.9.1.3 Effect of Application Measurements on the Algorithm Functionality

The correct detection of deviated nodes depends on the degree of coverage required in the neighbourhood. This was tested by conducting experiments with different degrees of coverage of temperature and light intensity sensors. Obtaining light intensity measurements was more challenging due to their high degree of change in the field. If the *WSN* coverage was low, nodes released many wrong warning messages even if the threshold was set at a high value. Figure 8-18 depicts the algorithm's warning messages that were released with the algorithm working with a static threshold. The figure shows a change in the level of measurements of light intensity especially between the

nodes at events larger than 300. The figure also shows an increase in the warning numbers released by the algorithm as deviations between the nodes increased. This was solved by adding a dynamic threshold that would set its value depending on the deployment of nodes and the data validation tests set for each of the detected deviations.

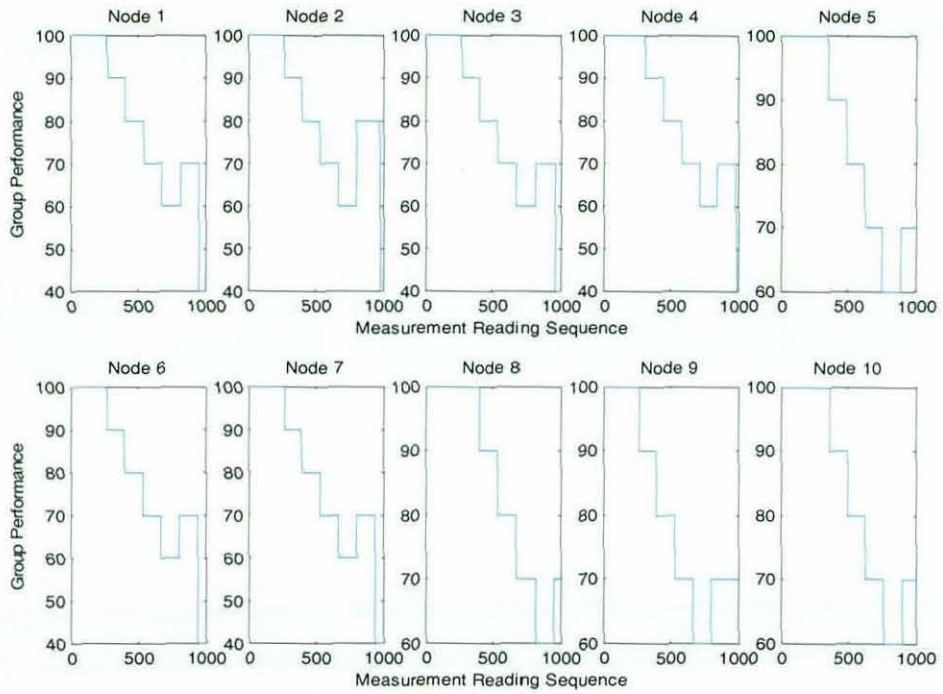

Figure 8-17. Network Performance Measurements with Nodes in a Group

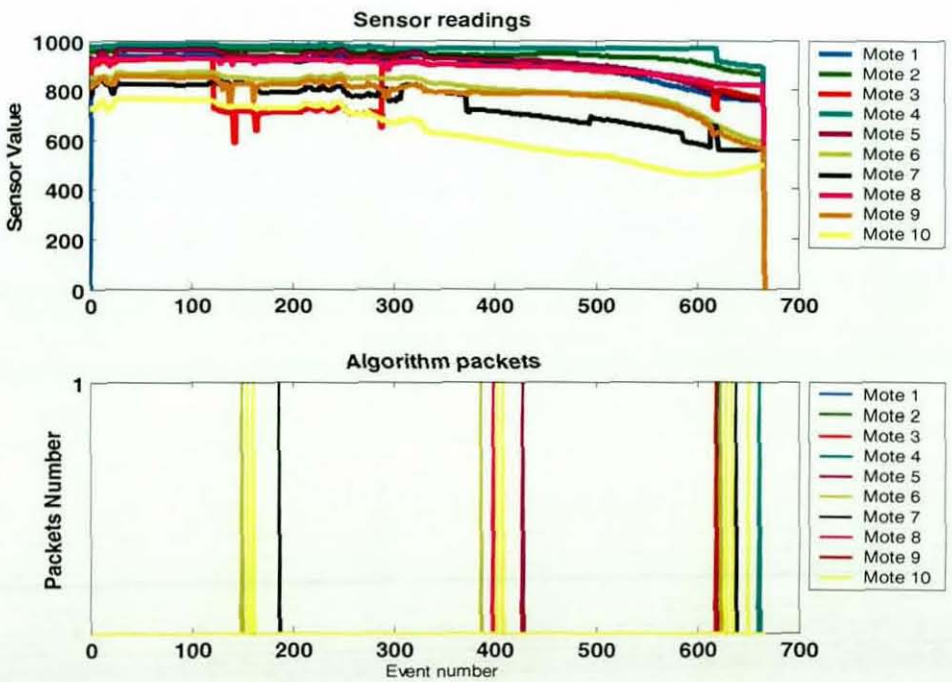

Figure 8-18. Experimental Results of Diversion Detection: Light Intensity Measurements

8.9.1.4 Effect of Thresholds on the Algorithm Functionality

Dynamic threshold settings are affected by changes received in the neighbour measurements and their losses at the threshold set up time, as can be seen in Table H-1 in Appendix H. Figure 8-19 shows the changes in the dynamic threshold standard deviation between neighbours versus the number of neighbours in the neighbourhood. It illustrates that the threshold standard deviation between neighbours increases as the number of nodes increases. This is due to the effect of losses and the degree of precision between neighbour node measurements. If this set percentage of dynamic threshold increases from the calculated neighbourhood median, the standard deviation between nodes in the neighbourhood increases up to a certain value and then starts to decrease, as shown in Figure 8-20.

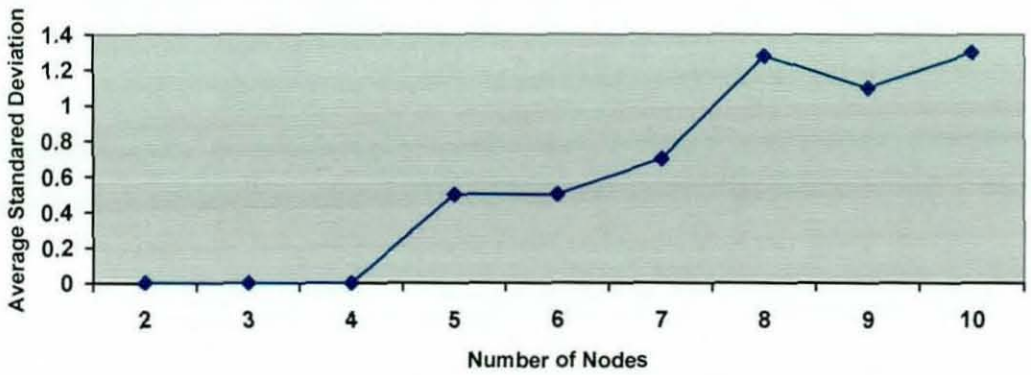

Figure 8-19. Average Standard Deviation of Neighbour Threshold Settings for Light Intensity Measurements

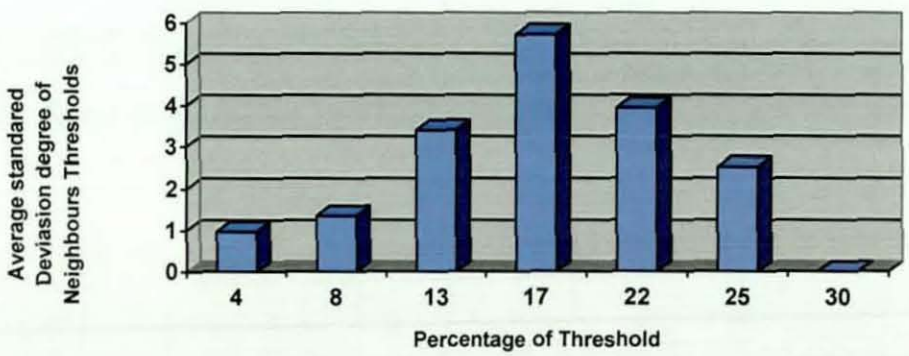

Figure 8-20. Standard Deviation between Neighbour Threshold Values with Different Percentage Settings for Light Intensity Measurements

To reduce the effect of these instantaneous changes in neighbour readings, the median of neighbours' readings were calculated. The differences between each reading and the calculated median were then multiplied by a factor depending on the detection level and the node coverage required. The results of this approach can be seen in Table H-2 in Appendix H where its value is more stable and the value hardly affected by losses or instantaneous changes in neighbours' readings.

Figure 8-21 shows the precise changes between neighbour nodes with a dynamic threshold set at 33% from the calculated median. The figure illustrates the linear change in the precision of the values of the neighbour thresholds up to a 5th neighbour node; it then remains stable as the number of neighbours increases.

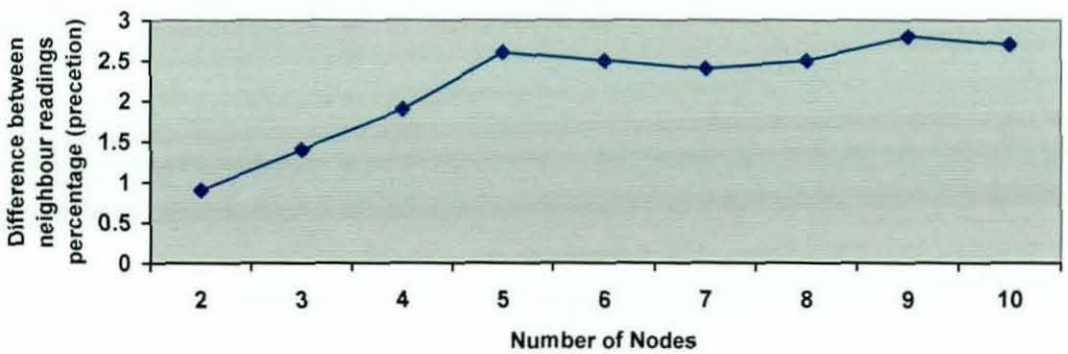

Figure 8-21. Precision Percentage between Neighbour Nodes in Light Measurement Experiments

These experiments showed that, for each type of measurement, different sets of threshold values were required due to the speed of change in the phenomenon characteristics. For example, with simulation experiments, the value of the optimal detecting deviation threshold for temperature measurements was 7% from the median, while empirical experiments showed that 30% is the optimal detecting deviation threshold for light intensity.

8.9.1.5 Effect of the Number of Faults on the Algorithm Functionality

As mentioned in Chapter 4, the total number of faulty neighbours affects the algorithm calculations since it uses a simple median voting technique. For

example, Figure 8-22 depicts the percentage of the algorithm detection, the positive and negative deviated nodes falsely detected by the algorithm versus the different percentages of faulty neighbour nodes for both method 1 and 2 implementations. The figure shows that, for method 1 implementations, if the deviated neighbour nodes number less than 50%, the algorithm detected all faulty nodes. As the percentage of faulty neighbours increases above 50%, the algorithm detects only 20% of faulty nodes along with a sharp increase in negative and positive false detections (i.e. these percentages depend on the number of nodes, the degree of deviation, losses, and the threshold). The Method 2 implementation, on the other hand, detects only 40% of the deviated nodes if the percentage of faulty nodes is 50%. (This is due to the lower reporting rate.) This reduces to 30% when the total percentage of faulty nodes reaches 60%, as shown in Figure 8-22.

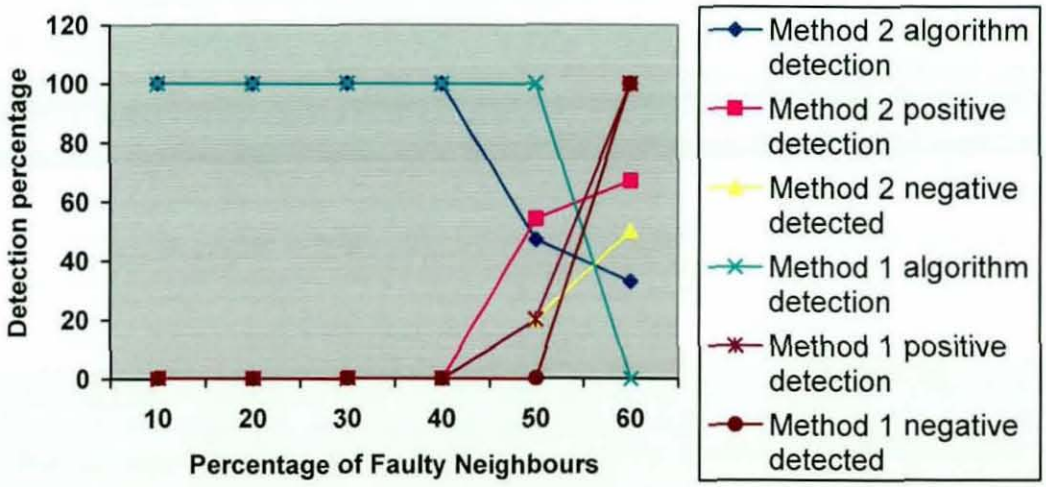

Figure 8-22. Algorithm Warning Message Exchange with Different Numbers of Faulty Neighbours

The number of messages released by the algorithm will increase linearly as the percentage of faulty deviated neighbour nodes increases, as shown in Figure 8-23. When the percentage of faulty deviated neighbour nodes reaches 50% in method 1 and 40% in method 2, there will be a sharp drop in correct detection and an increase in algorithm positive and negative false detection.

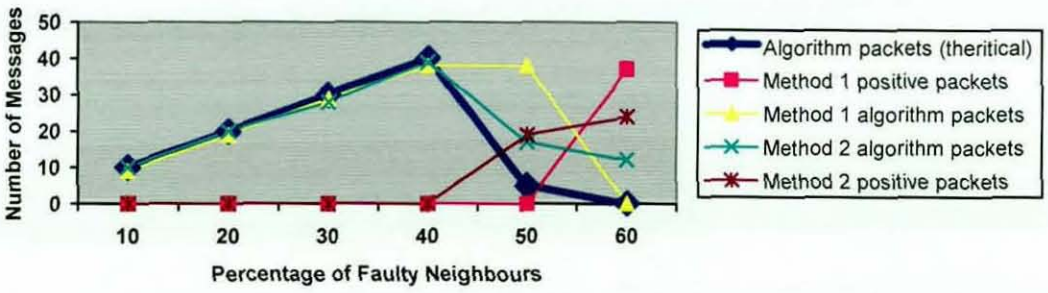

Figure 8-23. Number of Warning Messages versus Number of Faulty Nodes in 10 Neighbour Scenarios

These detections percentage and the number of algorithm messages that are released change with different sets of detection threshold values. Figures 8-24 to 8-26 show the percentage of detections of the algorithm, positive algorithm false warning packets and negative algorithm false versus different percentages of faulty deviated neighbours under different threshold values from the calculated neighbourhood median (i.e. method 1 implementation). The figures show that for up to 20% of deviated neighbour nodes, the number of warning packets released from the algorithm were similar. When the percentage of deviated neighbours increased above 20%, the threshold with a 30% value from the median did not detect some of the deviated faulty nodes. With 30% faulty nodes, the 4% threshold started to release positive false detections. Finally, with 40% deviated neighbours, all set thresholds, except 25% and 30%, showed a sharp reduction in the number of correct warning messages and an increase in positive false warning messages. (It is possible to optimise the trade off between positive false, negative false and selected threshold using the concept adapted by Patton in [124].)

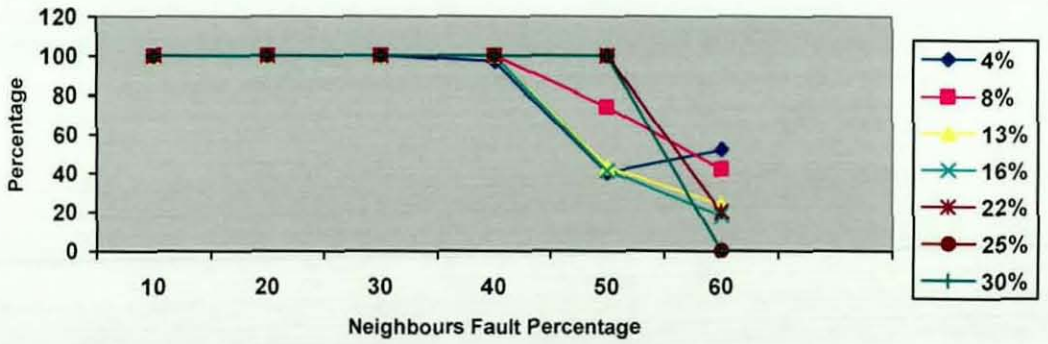

Figure 8-24. Percentage of Correct Warning Messages

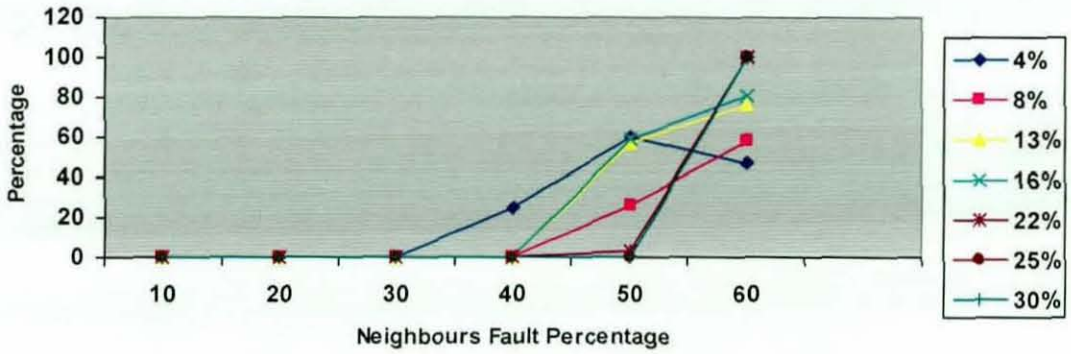

Figure 8-25. Percentage of Positive Warning Messages

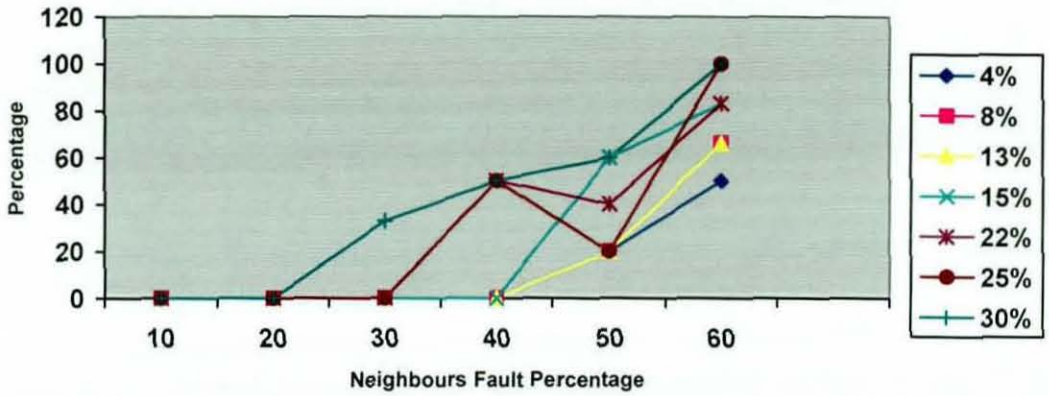

Figure 8-26. Percentage of Undetected Faults

These experiments indicated that changes occur in the detection if the threshold value changes and show an increase in the positive and negative false detection as the number of deviated nodes increases. Also, these experiments showed that the best threshold value for light intensity measurement is a value of 30% from the calculated median. This differs from the optimal value detected in the simulations for temperature measurements where it was 7% from the calculated median. This is due to the higher speed of change in the characteristics of light intensity as, with this, the detection threshold should be bigger.

On the other hand, tracking node aliveness is independent of the number of dead neighbours. This is shown by the experiments where the algorithm released dead warning packets even if all the neighbours were dead. These experiments show that, with 40% dead neighbours for Method 1, and with 60% for Method 2, there were negative false detections but no positive false

detections, as shown in Figure 8-27. However, Figure 8-28 shows that the number of released messages increased exponentially with up to 40% dead neighbour nodes; it then decreased linearly for both methods. In addition, these experiments showed that the dead node warning messages faced losses and these not only depended on the network configuration but also on the number of dead nodes at any one time. This result was not detected in the simulation experiments but may be due to the difference in speed between logical and physical operations (i.e. processing and transmission) and the node sleeping time. To solve this problem, more memory was added to store warning reports until they were sent. However, although this solution works up to 40% of nodes are faulty, after this, any additional memory space is without advantage. Another solution to this problem is to increase the wake-up period but this causes more power consumption. The last solution tested such that the algorithm reported the detection of dead nodes for a certain time then stopped. In this way, only a small number of nodes needed to be reported in the time interval.

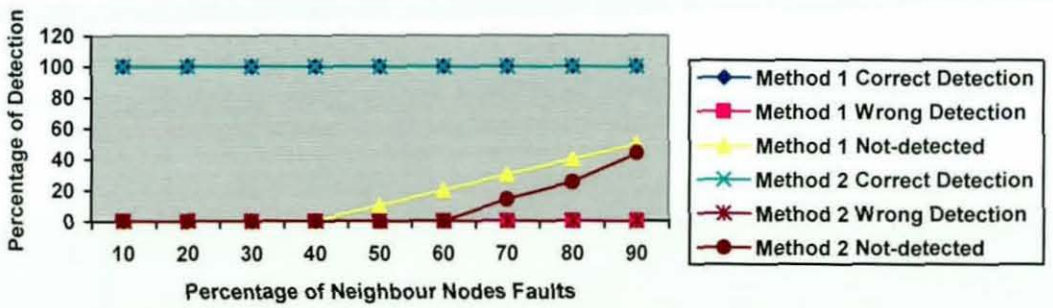

Figure 8-27. Reporting Messages for Dead Neighbour Nodes

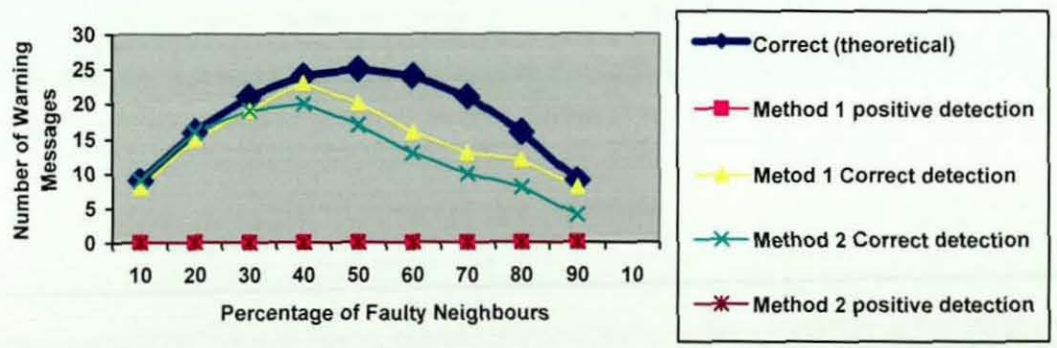

Figure 8-28. Number of Dead Warning Messages Reported for 10 Nodes Neighbours

8.9.1.6 Effect of Number of Neighbours on Algorithm Functionality

As discussed in Chapters 4 and 5, the number of neighbour nodes not only increases the confidence of detection but also increases both losses and energy consumption. Receiving packets from only one neighbour in the algorithm not only reduces the detection confidence, but prevent the faulty node from being distinguished. As the number of neighbours increases from one, the confidence in the detection of faults increases, as discussed in Chapter 6. When the level of faulty neighbour nodes reaches 50%, confidence in the released warning messages reduces due to reduction in the weighted deviation value from the calculated median; as discussed in Chapter 7. Figure 8-29 illustrates the percentage of detection versus the number of faults in neighbour nodes with different numbers of nodes in the neighbourhood. The figure shows that the percentage of correct deductions is the same for each consecutive even and odd number of nodes (as summarised in Table 8-5). For example, if there are 6 or 7 neighbour nodes, the maximum number of faulty nodes that the neighbourhood can detect correctly is 3.

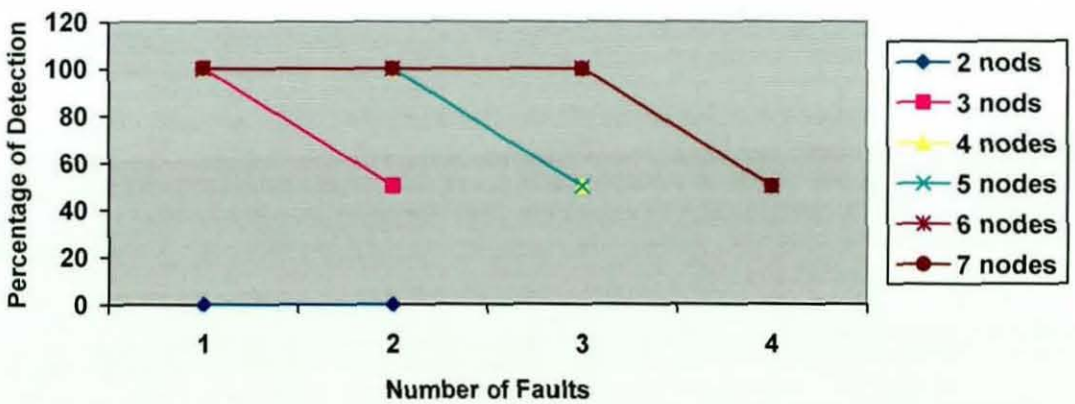

Figure 8-29. Percentage of Fault Detection versus Number of Faults

Number of faults	1	2	3	4	5	6
Number of neighbors	2,3	4,5	6,7	8,9	10,11	12,13

Table 8-5. Number of Faulty Neighbours versus Number of Neighbours

8.9.2 Network Configuration (Multi-hop)

Network experiments were conducted to test the algorithm functionality under multi-hop and highly dynamic topology configurations. These experiments used 13 Mica2 sensors, measuring temperature, distributed in an area of about 4mX5m (as shown in Figure 8-30). The nodes were programmed with an output power of -20 dBm and had top bent antennae to limit their communication range. In this configuration, the nodes were divided into two groups which overlapped in an area between them; thus, some of the nodes around the edge could not hear or communicate with each other (as shown in Figure 8-31). Moreover, this configuration forced the topology to be highly dynamic. This leads nodes to miss hearing each other and frequently change their multi-hop routing parents in the sink. These experiments used Mica2 nodes attached to a MIB510 programming board as a base station connected to a computer serial port. A snooping node was also added to the network with its power programmed at 5 dBm and a standard antenna. The function of this node was to monitor all the packets that were exchanged, including the network routing packets.

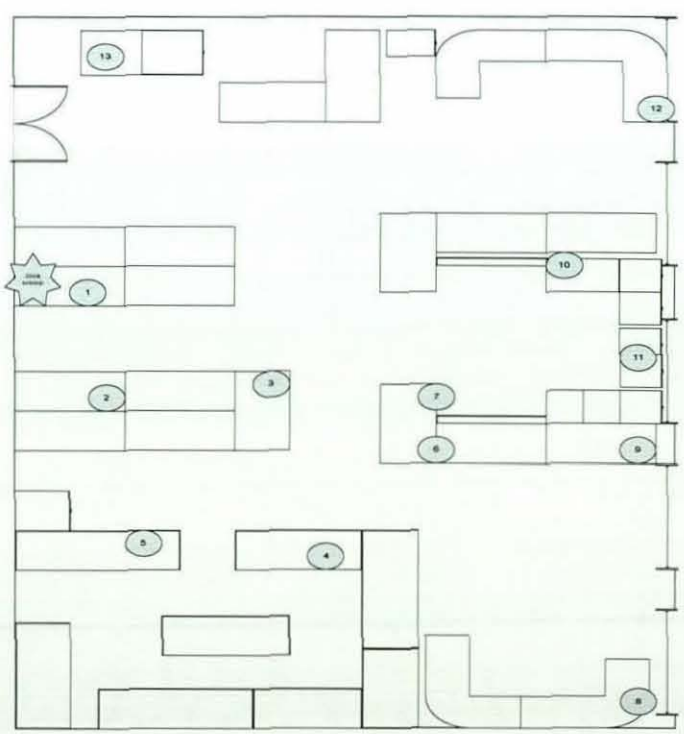

Figure 8-30. Physical Distribution of Nodes in the High Speed Network Research Group at Loughborough University

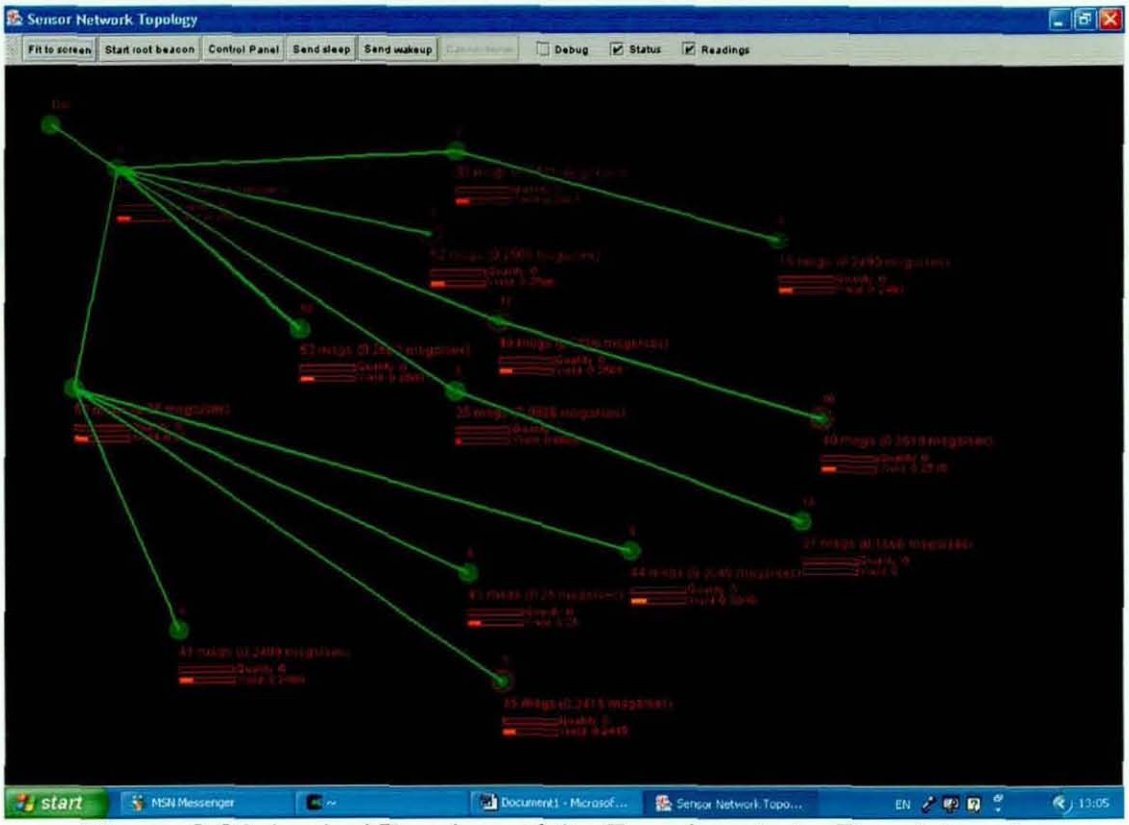

Figure 8-31. Logical Topology of the Experiment at a Time Interval

The metrics used to evaluate the results were, firstly, the percentage of incorrectly released dead node warnings. This is the ratio of the number of false dead node detections released by the algorithm as opposed to the total number of packets released by the application. This indicates the impact of high network dynamics on the incorrect detection of neighbour node aliveness. The second metric was the percentage of *'NO-FAULT-EVIDENCE'* messages released by the algorithm, which is the ratio of the number of *'NO-FAULT-EVIDENCE'* messages to the total number of packets released by the application. This also indicates the impact of high network dynamics on the neighbours' passive tests of incorrect detections.

In general, these experiments tested the impact of the dead node window threshold, and monitoring window size on the detection of dead nodes and the number of warning messages released in a highly dynamic network. The algorithm parameters that were tested, as shown in Table 8-6, were changed in different experiments to check their impact on the performance of the network and the exchange of warning packets.

Window Type	Small Monitoring window	Big Monitoring Window	Stop Reporting Window
Diversion	120 seconds (70% threshold)	480 seconds(8 minutes)	1920 seconds(32 minutes)
Distortion	60 seconds (85% loss threshold and larger than 25% accuracy of the two nodes)	240 seconds(4 minutes)	960 seconds (16 minutes)
Dead	60 seconds	240 seconds(4 minutes)	960 seconds (16 minutes)

Table 8-6. Sizes of Monitoring Windows in the Experiments

8.9.2.1 Effect of Network Topology and Packet Losses on the Algorithm Functionality

Figure 8-32 plots the relationship between the percentage of detected and '*No_Fault_Evidence*' messages released from the algorithm for different application reporting rates[17]. The results of the experiments showed that at a 1 second reporting rate (a multi-hop protocol leads to congestion and an overflow of communication, as discussed in [15]), a large amount of wrong suspected dead warnings occurred (around 3.2% of the total network packet exchange in the application). Furthermore, a large number of '*No_Fault_Evidence*' replies were released from neighbour messages (i.e. around 0.5% of the total packets in the network application). Reducing the reporting rate to 2 seconds reduced the number of suspected dead messages; these decreased sharply to 0.5% of the total number of packets released by the network application. This happened alongside a reduction in '*No_Fault_Evidence*' messages which reached around 0.01% of the total number of packets released. Thus, the number of suspected dead messages was reduced to almost 0% when the reporting rate was adjusted to 1 minute, along with a decrease in 'No_Fault_Evidence' messages released from neighbours. When the reporting rate was increased to 30 minutes, a sharp increase occurred in the number of suspected dead and '*No_Fault_Evidence*' messages, as shown in the figure. In addition, Figure 8-32 shows that, by increasing the reporting rate above 1 minute, the number of '*No_Fault_Evidence*' messages increases so that it becomes higher than the

[17] Please note that reporting rate logs were used in the figure to plot these.

number of suspected dead messages. This is as a result of the size of the monitoring windows and the highly dynamic network topology.

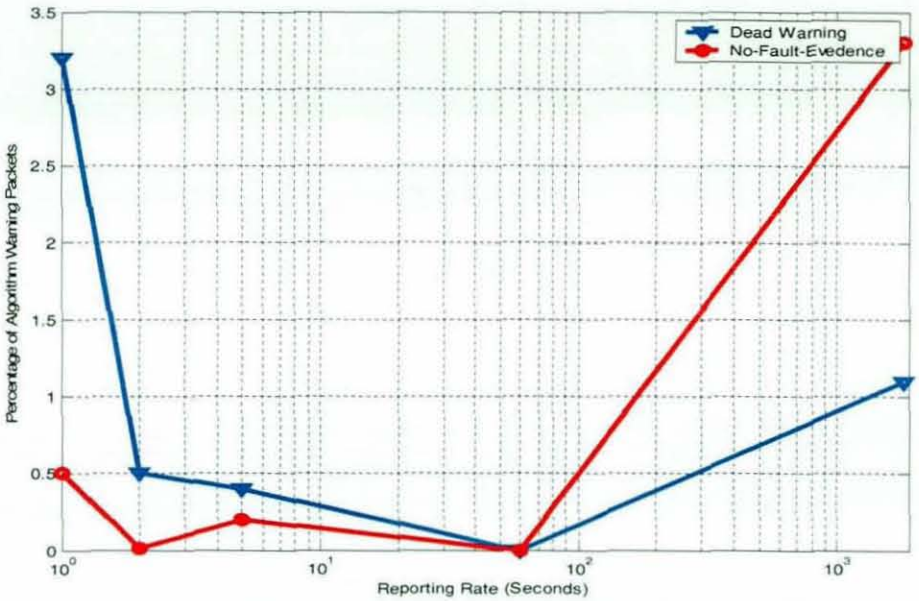

Figure 8-32. Changing Reporting Rates with the Percentage of Warning Messages Released with the Same Window Size

From these experiments, it can be concluded that dead node warnings will not disappear especially in a monitored network when the network connections are highly dynamic. To reduce the number of wrong suspected dead warning messages, different window sizes and combinations were tested, as shown in Table 8-7.

Window	Small windows	Small window size	Size of Big window	Number of small window at the group	Total monitoring window size
1	Linear increased	240 seconds (4 minutes)	3 groups	4-8-12	48 minutes
2	Exponential increased			8-12-16	64 minutes
3				10-14-18	72 minutes
4				14-16-20	80 minutes

Table 8-7. Size of Monitoring Windows

Figure 8-33 shows the relation between the percentage of correct, positive false detections by the algorithm, together with the negative false dead nodes for different sizes of large monitoring windows. The figure illustrates that, as

the big monitoring window size is increased, the confidence of detection of dead neighbour nodes increases, along with a decrease in the number of packets released by the algorithm. Although increasing window size will reduce the number of wrong messages, it also increases the response detection time and the probability of node failure occurring before releasing the warning message.

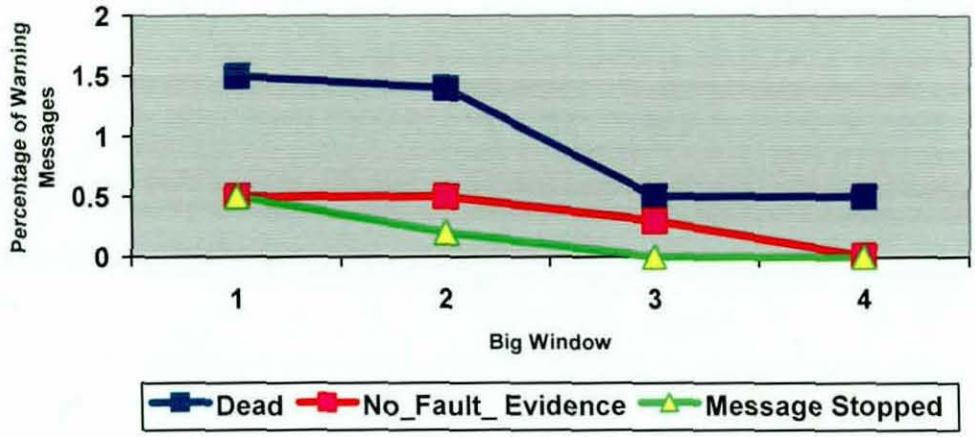

Figure 8- 33. Percentage of Warning Messages Released for Different Window Configurations

To solve this problem, the algorithm was programmed such that it would select the neighbours it would monitor; this selection depends on the amount of received packets. This configuration reduced the number of wrong packets reported by 80% and reduced 'No_Evidence_Fault' by 70%, as Figure 8-34 shows, but it also added additional complexity to the source code and its functionality. Moreover, there will be uncovered neighbour nodes in low density networks.

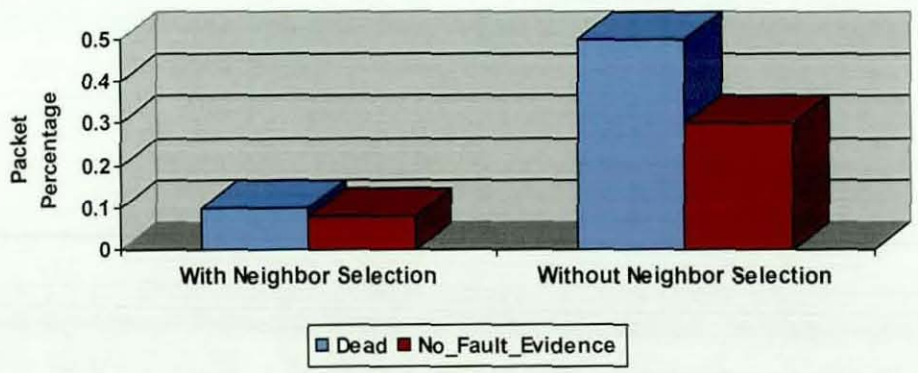

Figure 8-34. Number of Exchanged Warning Packets between Selected and non Selected Neighbour Nodes

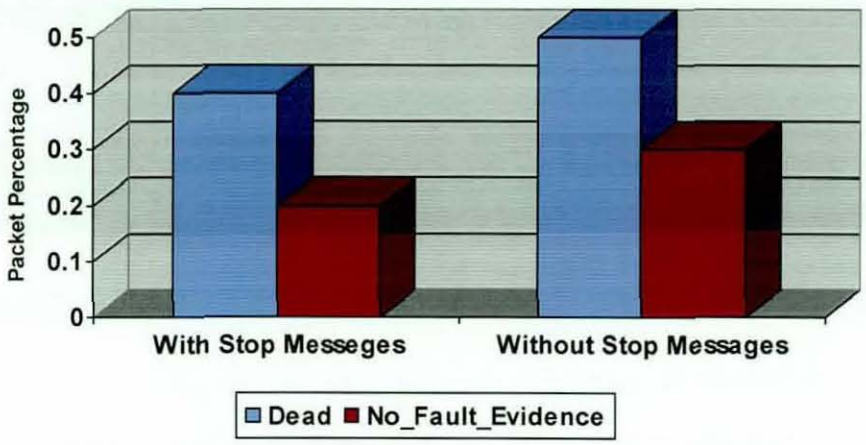

Figure 8-35. Number of Warning Messages after Using the Stop Message Indication

The proposed algorithm was modified to send warning messages concerning the detection of connectivity problems between neighbour nodes. This makes the algorithm stop reporting a suspected node if the node is detected as dead and after that 3 clear messages of the detection are released in a predefined monitoring window. Figure 8-35 plots comparisons between the percentages of the released dead and '*No_Fault_Evident*' messages in a neighbourhood with and without the modification covering connectivity problems. The figure shows that there is a reduction of 20% in the number of '*No_Fault_Evident*' messages as a result of a 34% reduction in the detection of dead packets.

In these experiments, there was repetition of some of the warning packets, as shown in Figure 8-36. These amounted to around 1.3% of the total number of network application packets when the reporting rate was 1 second; this was reduced to 0.3% when this reporting rate was increased to 2 seconds. When the reporting rate was reduced further, the number of these messages dropped to zero. This is may be due to differences between the logical and physical speed of software and wireless communications, and the effect of congestion in the network's shared medium (i.e. as network congestion increased, the losses increased and the number of repetitions increased). To rectify this problem, the source functionality was restricted between '*atomic*' commands and the size of tasks was shortened. This succeeded in removing the repetition of messages.

```
...
00 00 11 7D 0C 04 00 0B 00 44 03 01 00 FB 01 04 00
00 00 11 7D 0C 07 00 07 00 52 05 01 00 E8 01 00 00
00 00 0B 7D 0E 03 00 03 00 69 04 01 03 00 0A 00 08 04 00
00 00 11 7D 0C 05 00 05 00 6D 05 01 00 07 02 00 00
00 00 0B 7D 0E 03 00 03 00 69 04 01 03 00 0A 00 08 04 00
00 00 0B 7D 0E 03 00 03 00 69 04 01 03 00 0A 00 08 04 00
FF FF FA 7D 1C 03 00 03 00 6A 04 01 00 00 06 00 00 FB 02 00 EE 0D 00 E6 04 00 D7 0C 00 A5 05 00 9B
00 00 11 7D 0C 07 00 09 00 53 05 01 00 F2 01 07 00
...
```

Figure 8-36. Repetition of Warning Message Reporting

The experiments showed that high losses impact on the algorithm functionality if the algorithm works in a highly congested and high loss medium. The aliveness tracking analysis will send incorrect warning messages of suspected dead nodes and there will be a reply to these messages from neighbours; i.e. *'No_Fault_Evident'* messages. These messages cannot be avoided especially in a highly dynamic topology but they can be reduced. These messages were used in the algorithm as an indication of the network stability and, when the network is unstable, there will be a high level of exchange of wrong warning and *'No_Fault_Evident'* messages whereas, if the network is stable, there will be no or very few wrong messages of suspected dead nodes. This was tested by removing two nodes from their original places (i.e. nodes 5 and 7 in Figure 8-30) after the network was functioning for seven hours without any warning messages. Moving nodes 5 and 7 caused an exchange of suspected dead warning messages and instability in the connection between other nodes in the network. After 45 minutes, the two nodes were returned to their original places and, after two monitoring periods, warning messages stopped in the area the two nodes were shifted to. Also, the algorithm uses these loss statistics to calculate the criticality of neighbour readings that are not received (i.e. distortion) on the sending of phenomenon measurements to the sink, as discussed in Chapter 4.

Moreover, the experiments show that warning messages faced some losses while they were being sent to the sink. This was detected from the presence of *'No_Fault_Evidence'* messages that reached the base station without a warning message being received and increments in the warning counter that showed a missing packet. If the algorithm works with the model of releasing

one warning message for a group at a monitoring time interval (this was built into the algorithm to reduce energy consumption), some messages will not reach the sink. However, when each node is allowed to send at least a fault detection, a message reaches the sink. This can be solved by releasing the total number of detections. This will convey that there is degradation in the neighbourhood performance but there will be no need to know which node has degraded the performance of the group since the total indication of degradation is there. This can also be solved by allowing nodes in the neighbourhood to release the same detection message a few times (3 times, for example) to reduce the probability of losing a warning message.

8.9.2.2 Fault Timing

The experiments showed that the detection of faults depends totally on the time the fault occurs and its position in the big and small monitoring windows; this may cause a delay in detection, as discussed in Chapter 4.

8.9.2.3 Effect of Battery Depletion

To check the effect of battery depletion, nodes in the network worked for 8 days continuously until there was a collapse due to battery depletion; this caused a sudden shutdown in the entire network. The nodes in the overlap area and those near the sink stopped sending their readings before the others, such as node 1 shown in Figure 8-30. Before this happened, these nodes reduced their reporting rate then stopped completely.

A sudden drop in the node group was observed due to a drop in voltage; this does not allow the algorithm to send dead node warnings. The same observation was made in the habitat monitoring application designed by Szewczk *et al.* in [125]. He related this to the rapid exhaustion of nodes sharing a multi-hop network. He assumed that the drop in the battery power was rapid and not recordable. Also, Zhao, in [14], faced the same sudden stop in the network functionality; he related it to the application's continuous and equal power consumption which drained the battery of neighbour nodes equally.

To overcome this problem, a warning packet was added to be released in the *'AMPromiscuous.nc'* component, as shown in Figure 8-37, so that this would be sent before the node stops participating in the network. This allowed the algorithm to use the voltage level at the node as an indication. If this level is less than the minimum value set by the sensor manufacturer, the node sends a dead packet to the sink and its neighbours consider it to be dead.

```
...
  command result_t Control.stop() {
  result_t ok1,ok2,ok3;
  if (state) return FALSE;
  ok1 = call UARTControl.stop();
  ok2 = call RadioControl.stop();
  ok3 = call ActivityTimer.stop();
call NeighborsTable.SenFaltReport(TOS_LOCAL_ADDRESS, TOS_LOCAL_ADDRESS,3,3);
  call PowerManagement.adjustPower();
  return rcombine3(ok1, ok2, ok3);
}
...
```

Figure 8-37. Code for Releasing a Warning Message Before a Node Stops Sending Packets

8.9.2.3 Effect of Reporting Rate

As discussed in Chapter 4, the algorithm's detection confidence can be changed by varying the size of the monitoring window. However, if the reporting rate of the network application is low, the longer monitoring windows reduce the algorithm's response in terms of detecting and isolating faults. (This increases the impact of the faults on the network's functionality.) On the other hand, if the size of the monitoring windows is reduced, the probability of the algorithm reporting wrongly will increase, as discussed in Chapter 4. In addition, the empirical experiments showed that there would also be an increase in the number of *'No_Fault_Evidence'* messages. For example, in the conducted experiments, when the application reporting rate was every 30 minutes, there were numerous warnings of deviations and *'No_Fault_Evidence'* messages when one sample size monitoring windows were used. The number of these messages was reduced, however, when data validation tests, such as using a number of similar readings which were close to the calculated median, were carried out. Unfortunately, nodes that did not have more than one neighbour in the neighbourhood were not able to send a warning message. This occurred even if these nodes detected a fault.

(This was because they did not satisfy the conditions of the data validation tests.)

8.10 Method 2 Algorithm Implementation

As discussed in Chapter 4, the Method 2 implementation of the algorithm offers the possibility of a hybrid application network functioning with different application reporting rates. The Method 1 implementation, on the other hand, failed to detect node faults in the application, such as those that were event-driven, if the fault value was less than the assigned event-reporting threshold. For example, Figure 8-38 shows the event-driven implementation for Method 2 with a reporting threshold set at an *ACD* raw sensor measurement of 990 for light intensity. The figure shows that the algorithm detected the faulty nodes, even when the measurements did not exceed the assigned reporting threshold. In order to test the reliability of Method 2, the Method 1 implementation was used under the same experimental configuration but with a continuous reporting application: the same results were detected.

8.11 Algorithm Usage

The algorithm detection output can be used in three main ways: to provide direct packet readings, online visualisation of received packets, or for self-configuration.

8.11.1 User Level of Packet usage

Network users can check network degradation directly from the warning packets received at the sink. Figure 8-6 shows one of the warning packets that was detected while empirical experiments were conducted using 10 neighbour nodes measuring light intensity. This packet indicates that node 1 suspects that node 3 is a faulty deviated node. This deviation degrades the neighbourhood performance by 40% (i.e. Number of Detections of Deviation ÷ Number of Readings). Moreover, the packet indicates that there is no detection of dead nodes in the neighbourhood.

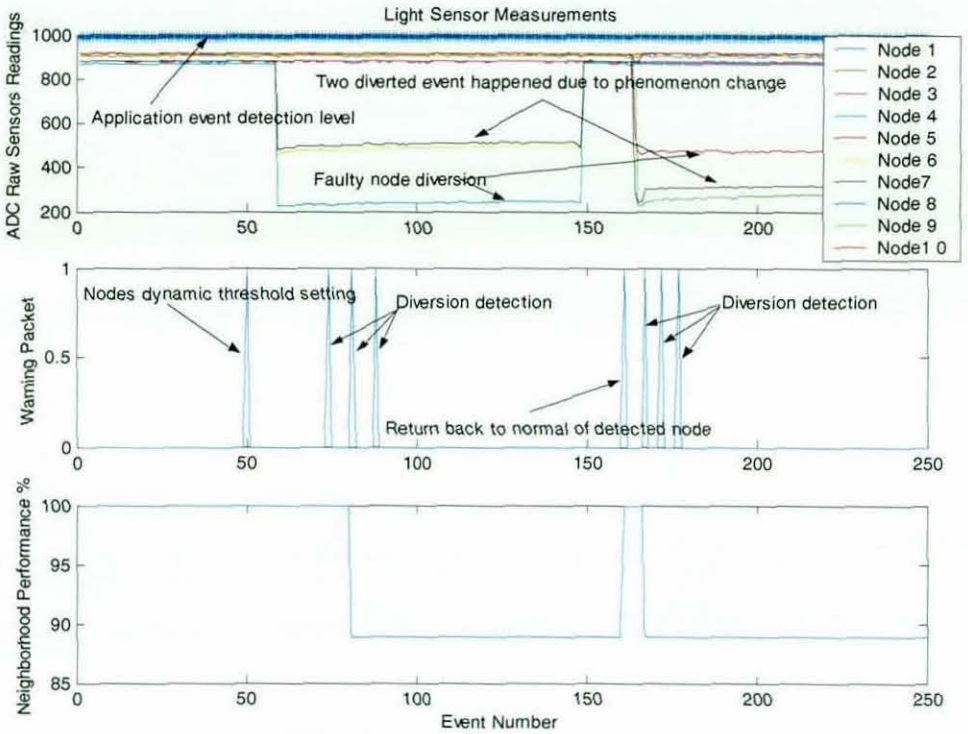

Figure 8-38. Detection of the Method 2 Algorithm in an Event-driven Application Set at 990 Event-Driven Reporting

8.11.2 User Level Visualisation

Warning packets collected from the algorithm can be used to visualise network functionality, as shown in Figures 8-17 and 8-38. This is achieved by equipping the sink with a visualisation tool that monitors the real-time network when receiving a warning packet from a node. After receiving the warning packet, the program starts a counter of size equal to the size of the monitoring window and, by the end, the program expects to receive another packet if the fault still exists. If the program receives the packet before the end of this period, it will visualise the packet contents. Otherwise, it returns the monitored performance of the neighbourhood to the performance value that was current before this fault occurred. If a 'Fault_Message_Stop' packet is received, the program is going to continue visualising the last calculated network performance level for that neighbourhood. This will continue until the reception of a 'Fault_Clear' packet of the same fault where the program re-calculates the effect of fault clear on the neighbourhood performance. The main problem with this method is its high sensitivity to any lost warning messages.

8.11.3 Self-Configuration

The algorithm can be used to provide input to self-configuration, routing, collaboration and other protocols. This would reduce the neighbourhood's dependency on a suspected faulty node, thus reducing its impact on the network functionality.

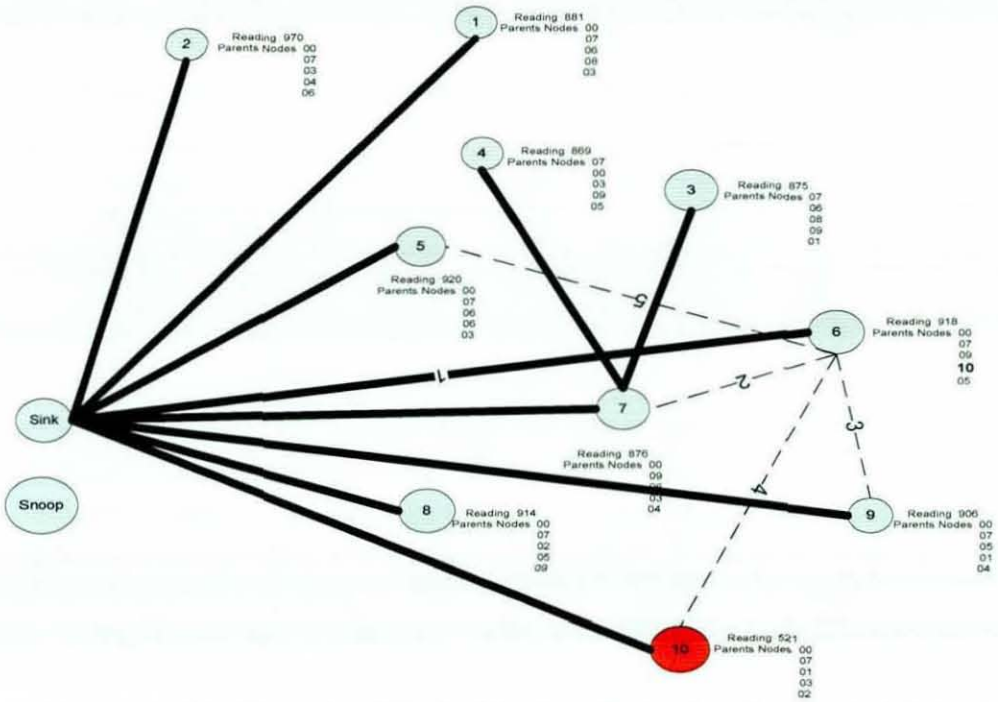

Figure 8-39. Parents of Node 6 before Detecting the Node 10 Deviation

In order to test this, the *TinyOS* multi-hop routing protocol was modified so that the output of the algorithm could be used as metrics for re-configurating the network. Figures 8-39 and 8-40 show a snapshot of the hexadecimal output of the *'Surge'* multi-hop network routing table (see Appendix I) for an experiment using a distribution of 10 nodes to obtain light intensity measurements. When node 10 was forced to deviate from its neighbours, the algorithm detected this deviation and checked the contents of the multi-hop table by searching for node 10 in the routing table. (Thus, the algorithm searched node 6, which had nodes 0, 7, 9, 10, 5 in its routing table.) Then the node recalculated its routing table so that node 10 was given the lowest priority for selection in that routing table. Node 6 would then select 0, 7, 9, 5, 1.

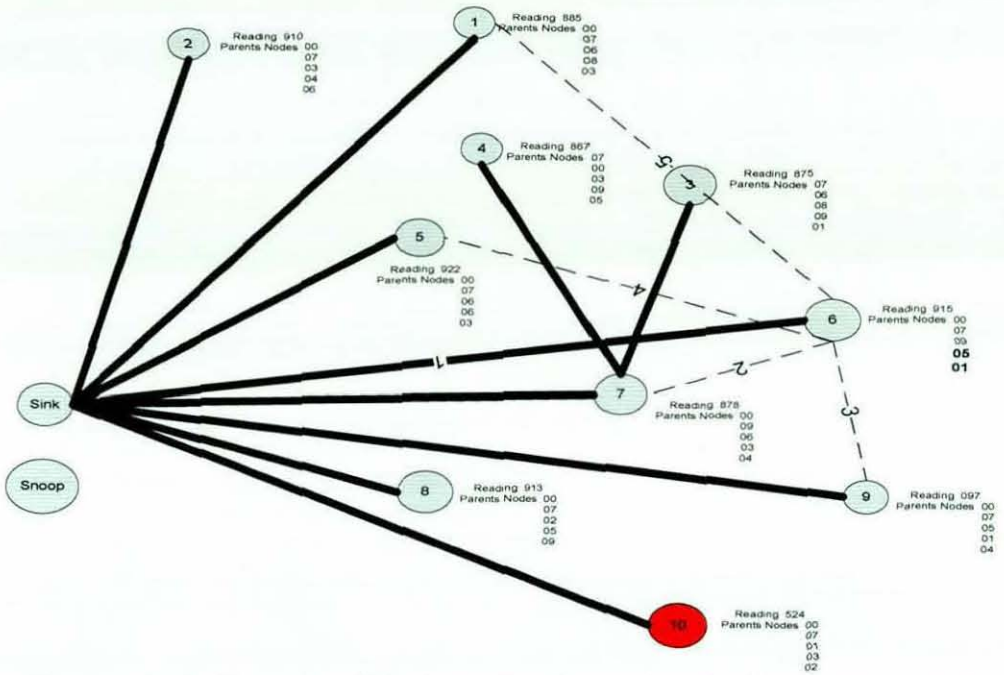

Figure 8-40. Parents of Node 6 after Detecting the Node 10 Deviation and Recalculating the Multi-hop Routing Neighbours

8.12 Summary

The empirical experiments showed the proposed algorithm's high success in detecting of *WSN* faults. These experiments detected almost the same behaviour in the algorithm that been detected in the simulation experiments and that were discussed in Chapter 7. The only difference seen was the negative false detection in dead neighbour nodes as the number of dead nodes increased.

As detected from the empirical experiments and discussed in this chapter, the suspected dead warning messages were affected by losses and, in order to reduce this effect, the aliveness monitoring window should be increased to a higher level of confidence (as discussed in Chapter 4). However, latency will be a tradeoff in reporting dead messages if this is done. On the other hand, the experiments showed that these losses do not affect the faulty deviation analyses since the algorithm monitoring window threshold depends on the percentage of received packets.

The experiments also showed that there is a tradeoff between detection accuracy and response time for fault detection as the reporting rate becomes low. Thus, as the reporting rate reduces, the algorithm should rely more on data validation tests for its detection; window size should also be reduced.

Finally, these experiments showed that allowing only one node in the neighbourhood to send warning messages sometime means that warning messages are lost and the fault is not reported to the sink. This affects the message exchange especially if the algorithm is used to report changes to the sink. This can be solved by increasing the number of reporting neighbour nodes to reduce the probability of losing the warning messages.

Chapter 9 Conclusions and Future Work

9.1 Introduction

Although the characteristics of *WSNs* allow them to be used in a wide range of applications, they reduce their immunity to external/internal interference. There is also an increased probability of a sensor node measurement of a phenomenon deviating from the actual value. These deviations affect the quality and quantity of the data collected as a result of their impact on the network functionality. This is because they may affect routing, data gathering, the reporting rate and data processing, which all reduce the network performance and increase its' usage of resources.

In some of the practical deployments; such as [62], [126]; the analysis of collected network data showed a reduction in data quality that reached to 49% caused by these deviations. As a result of this reduction, the network needed, in some cases, to be re-deployed again to collect the required data since some of the data collected by the network were meaningless. In addition, these analyses indicate that there is a possibility of improving the deployed network functionality up to 51%; in term of resources usage and collected data quality; if real-time monitoring tools that detect the deviation and enable network user or network protocol to reconfigure and solve the problem. So, to ensure the reliability of the collected data and of the network itself, a tool is required to monitor and detect these deviations before they have a high impact on network functionality.

This thesis has proposed a performance monitoring tool that collects passively three basic metrics used in all types of *WSN* application: neighbour node identification, packet loss and neighbour measurements. From these, it calculates events that indicate states of change in the network functionality and in the accuracy of the network collected data.

The algorithm proposed has been tested using both simulation and empirical experiments in order to satisfy both scalable and real-time requirements. These experiments showed a high level of reliability in detecting changes in both the network status and the accuracy of the data collected by the network.

9.2 Contribution of this Thesis

The major contribution of this thesis is the development of the distributed algorithm that monitors *WSN* performance online with a low resource usage and a very low impact on the network lifetime. The simulation experiments, for example, showed that, for an average of 16 neighbours, the algorithm reduced the *'Surge'* application network lifetime by 0.01% (i.e. 0.03 of a day) when 50% of nodes were permanently deviated, as discussed in Chapter 6.

The second contribution made by this research is the identification of simple metrics that can be extracted from most *WSN* applications (i.e. packet losses, neighbour identification and neighbour measurements). These metrics are used in an application to guide routing, to decide on data gathering points, to control the reporting rate of nodes and for in-network collaboration. From these metrics, the proposed algorithm calculates the relationships between high and low network levels and detects the status of neighbourhood nodes (i.e. node malfunction, node aliveness, neighbourhood malfunction, neighbourhood data accuracy, energy usage, and neighbourhood connectivity), as discussed in Chapter 4. Most research to date has concentrated only on low-level network parameters evaluations; i.e. packet loss; for indicating the health of the network. This is due to the difficulty of monitoring sensor health through high-level information; i.e. nodes measurements[18]. This was solved in the proposed algorithm by using a neighbourhood median to estimate norm measurement. A difficulty in distinguishing between changes in the phenomenon, the environment and the fault was also discovered. This was solved in the proposed algorithm by using neighbourhood voting and a threshold scheme for distinguishing the type of change; as discussed in Chapter 4.

The third contribution of the study is the effect of the algorithm location on the implementation code complexity and functionality, as discussed in Chapter 5. (This is believed to be the first time any researcher has studied this aspect).

[18] This is because of the unavailability of ground truth for phenomenon characteristics and the highly complex and expensive nature of the methods that predict absolute references for phenomenon characteristics (as discussed in Chapter 4).

The final major contribution of this thesis is how best to use a combination of the three techniques (i.e. analytical, simulation and empirical) for testing the functionalities of new *WSN* algorithms. Most previous research has concentrated on environmental dependence when evaluating algorithms but this thesis tested each aspect of the algorithm using the three methods and compared the results with each other in order to overcome the limitations of each individual method. The experiments showed that some scenarios work successfully using certain methods while others failed as a result of the complexity of the network and its functionality. For example, releasing warning packets to indicate dead nodes was very successful in the simulation method while, using the empirical method, after 40% of the neighbourhood nodes were shown to be dead, the messages reduced in number and some were not reported. This is discussed in Chapter 8. These experiments showed that a simulation approach is most suitable for testing an algorithm's resource usage, the impact of losses, and the algorithm's probability of detection. On the other hand, empirical methods are more suitable for testing packet release behaviour and the effect of the algorithm complexity on network node behaviour.

9.3 Characteristics of the Algorithm

The proposed solution has adapted a performance monitoring technique to detect changes in the network's health, as discussed in Chapter 3. This performance monitoring technique uses fault management because failures in *WSNs* are a common event, unlike in a traditional network where these are usually the exception, due to limited resources and the wireless communication. A large variety of these faults may impact badly on the quality and quantity of the data collected by the network and may affect the network's functionality (because these data are used in reconfiguring and reorganising the network).

The proposed algorithm uses passive monitoring in collecting its metrics and analyses them to detect several events that indicate changes in the status of

the network nodes. These metrics are node neighbours, neighbour packet losses, and neighbour measurements. Selected metrics are used in all applications to self-organise or configure the network and to control collaboration between network nodes in the collection and communication of data.

Moreover, the algorithm uses a majority vote method among neighbours to estimate the norm measurement of the phenomenon. This is different from most research that has used the monitoring node measurements as a reference or has used a complex method to predict phenomenon measurement that needs special resource nodes; as discussed in Chapters 3, 4 and 6.

The algorithm functionality depends on deviations from assigned threshold values (i.e. the value that is assigned depends on the network coverage/connectivity/accuracy required, and the sensor characteristics, as discussed in Chapter 4). It also depends on the comparison between changes in high and low network levels to detect changes that may affect the quality and quantity of the collected data.

Detection confidence of the proposed algorithm depends on monitoring window size and data validation tests whose values are controlled by the application reporting rate, the application tolerance to the change, and the required response time, as discussed in Chapter 4.

To ensure low power consumption in the proposed algorithm, the transceiver is only used when an event is detected within a given time or when a monitoring node does not agree with a warning message released from its neighbours.

9.4 Difficulties Faced While Carrying Out the Study

Due to the lack of literature and research that exists on systematic performance measurement and monitoring in *WSNs*, various types of *WSN*

applications and network protocols, and the limited availability of real network data measurements, it was very difficult to understand network behaviour in special terms, the nature of collaboration, and loss tolerance. The performance metrics used in the literature that was available concentrated on comparing the resource usage and functionality of different protocols in order to select from them the best for a particular application. These metrics can be used only in simulation or on small testbeds due to their high resource usage, as discussed in Chapter 3, and so are not suitable to be used in real-time, large-scale network monitoring. The only metrics found in the literature that was used for real-time monitoring of network health were residual energy and the connectivity map. However, these metrics, apart from their relatively high power consumption and the need for a central controller for management, do not, on their own, give a good indication of network health, as discussed in Chapter 3. Because of this, the research reviewed the studies that were available concerning traditional wired sensors and how they are measuring performance. Then it concentrated on the occurrence of faults in sensors and *WSNs*. This was followed by reviewing the literature describing the impact of different types of fault on the network's functionality and the accuracy of the data collected by the network.

The second problem faced in the study was the lack of availability of simulation software that would simulate Wireless Sensor Network behaviour at both high and low network levels. Most of the simulations, such as the *NS2* simulator and its sensor networks model extension (i.e. *NRL*), simulate ad hoc network characteristics at low network levels. To simulate high network levels, *MATLAB* software was created with the ability to receive the readings from neighbour nodes and then to calculate the function of the algorithm using them, as discussed in Chapter 7. By integrating the results of the simulations, a good understanding of the network, data collection and the algorithm behaviour was achieved. The aspects that were not considered in these simulations were the impact of a dynamic topology on the algorithm and its functionality, the impact of the environment and the impact of the algorithm complexity. These aspects were studied by using empirical experiments on a Mica2 testbed, as discussed in Chapter 8.

Finally, the practical experiments were difficult to conduct and debug due to the event-driven nature of the operation system, and because the parameters to control any empirical implementations are challenging, as discussed in Chapter 8.

9.5 Discussion of the Results

The results from the conducted experiments can be summarised in six points:

9.5.1 Algorithm Performance

The experiments showed that the detection performance of the proposed algorithm depends on threshold value, loss percentage, the number of healthy readings per event, and the duration of the fault. As a result of this, the algorithm was designed so that it reduced the impact of these factors on its detection using a dynamic threshold that depends on the application accuracy requirements (as discussed in Chapters 4 and 8). It also used a design that included two monitoring windows that were concerned with the detection of temporary (the small window) and permanent (the large window) deviations. The values of the monitoring window thresholds, together with the percentage values of packet losses, were used to reduce the effect of these losses. In addition, data validation tests were used to reduce the response time of the detection and increase the confidence of the algorithm detection. This was discussed in Chapter 4.These design aspects allowed the algorithm to detect 64% to 97% of the deviated faulty data that affected the quality of the data collected in the network. The reason the other deviations were not detected was because of their duration and changes in the voting assumption of concurrence between neighbour measurements due to changes in the phenomenon or the environment characteristics, as discussed in Chapters 6 and 7. (It is worth to mension that these undetected deviations do not affect the functionality of network because of their short duration.)

Moreover, these experiments showed that deviation detection was reduced when the number of deviated measurements in the neighbourhood neared

50%. This caused positive false detections and increased the number of negative false, as discussed in Chapters 6, 7 and 8.

On the other hand, these experiments showed that the detection of aliveness depends on the percentage of losses (as discussed in Chapters 4, 6 and 7) and the number of dead nodes, as discussed in Chapter 8.

9.5.2 Resource Usage

The simulation experiments that were conducted showed that the proposed algorithm used low resources and energy (because it used common nodes parameters and reused resources used by other protocols, as shown in Chapter 6). This usage can be summarised as processing and *RAM* usage, and the required packet warning exchange (i.e. a maximum of 0.8% of the original processing consumption without the algorithm and with an average of 16 neighbours). The reduction in the algorithm's usage of resources affected the lifetime of an individual node by an average of 0.01% (tested on *'Surge'* application). This was discussed in Chapter 6.

9.5.3 Algorithm Limitations

The experiments showed that certain limitations may affect the algorithm detection: this depends on the number of neighbour nodes, the number of unhealthy readings received per event, and the degree of deviated neighbour readings received, as discussed in Chapter 6. Such effects may reduced by using monitoring windows and data validation, as discussed in Chapter 4. These control the effects of the limitations mentioned above and are tradeoffs with detection accuracy, detection confidence and positive/negative algorithm false detections, as discussed in Chapters 7 and 8.

9.5.4 Packet Loss Effects (Threshold, Released Packets, Detection)

The relation between packet losses and detection is complex due to the dependency on loss percentage, loss measurement, the number of healthy neighbour readings and the location of faults. The impact of packet losses can be reduced by using a window threshold depending on its percentage and by

using data validation tests. However, as packet losses increase, the detection confidence of the algorithm decreases. It takes longer for the algorithm to detect a fault, it may cause warning packets to be repeated, and the detection order may be different from the fault occurrence, as discussed in Chapter 7.

9.5.5 Location in the Application Code

The programming location of the algorithm in the application code is important because it controls the algorithm's complexity and influences modifications for different applications, as discussed in Chapter 5. This factor is important for any algorithm but has been ignored by other researchers.

9.5.6 Methodology of Evaluation

The experiments showed that, due to the complexity of *WSNs* and the challenges they present, any evaluation of an algorithm should be carried out using the three research methodologies, analytical, simulation and empirical. One is not enough on its own to achieve an accurate evaluation, as shown by the results offered in Chapters 6,7and 8.

9.6 Future Work

Future work will largely focus on three main factors: to extend the functionality of VMBA, to make it more efficient and to improve its usability. It is proposed that this could be achieved through four main activities.

The first is to study the effect of node mobility on the algorithm performance. Mobility is a very important factor that should be considered due to its impact on the network functionality. Basically, there are three types of mobility scenario available in *WSNs*. The first is when network nodes are stationary and the target is moving: e.g. applications that monitor a phenomenon to check the availability of an objective in an area, such as checking the entry of cars in a car park. The second is when nodes move toward/away from a target, such as sending sensors to a building when there is a fire to check the safest routes that a rescue team can take and the location of people in the

building. The third is when both the nodes and the monitored target are moving, such as when the network tracks the movement of an animal by using mobile sensors. Each of these applications has its own characteristics and challenges that can be used to reduce the impact of the algorithm on the lifetime of the nodes. In general, node mobility increases the event boundary effect that was discussed in Chapter 5. Moreover, it changes the level of correlation between nodes, the distance between nodes, and the number of neighbours within an area. It would be useful to study mobility by examining the parameters that are available and that are required in the network, the collaboration function between adjacent nodes, the handshaking operations between the nodes, and the correlation change as a function of movement. It also is interesting to test the effect of mobility on deviations that occur between neighbour nodes and it is intended to allow the algorithm dynamically to adjust its monitoring windows depending on mobility levels.

Secondly, the researcher intends to build an influence diagram (one that will depend on the confidence of network parameters) in order to reduce the detection response time and the effect of deviations on the network functionality. This will be achieved by building a decision-analytic model that will identify harmful deviations and their level of impact on the network protocol functionality. An influence diagram is important in networks with a low reporting rate and in low-density/low-coverage applications. This is because it will reduce uncertainty in the proposed detection algorithm by relating different available parameters to each other in a "cause and effect" manner. Several inference procedures and sensitivity analyses, based on this model, could be tested in order to illustrate how deviation detection impacts on sensor functionality or sensor failure. The problem with this technique is that it is complex and it is also application specific. Any study should try to reduce this complexity by relating the model's functionality to the expected type of fault and to the degradation in the network protocols, not to the application itself as is more usually done.

It is further proposed to test, in practical implementations, the functionality of the proposed algorithm and to study its performance in network applications

other than an environmental monitoring application, a tracking application, for example. These applications have their own characteristics that can be utilised to simplify the algorithm. In addition, each application has its own unique challenges that need to be considered when the algorithm is working. The simulation experiments in Chapter 5 showed a good level of detection regarding the health of the network when the functionality of the algorithm changed with the characteristics of the application. Unfortunately, due to the lack of availability to the researchers of data from real implementations, it is difficult to know how the algorithm will perform under real circumstances. A study to investigate the sensor data used by different applications to obtain a better understanding of the data models would be useful. This will answer the question regarding the parameters that can be used in order to reduce the impact of the algorithm on network node resources.

Finally, it is planned to extend the functionality of the algorithm to work as a *WSN* management tool and to compare its efficiency with one of the existing *WSN* management tools. At the moment, almost all of these management tools either work actively or proactively when the network is tested and/or maintained. This is done by constructing a maintenance tree between all the network nodes and the sink, so that the required information flows through it. This maintenance tree would either sends test packets from the sink or diagnosis packets from nodes, as explained in [63]. Our proposed algorithm uses a passive monitoring method and, when it detects a problem, it sends a limited number of warning packets during a predefined time before forcing the network to reconfigure itself. It is planned to test the functionality of the algorithm as a management tool by testing it with one of the well known *WSN* management tools (this would be selected later on) in order to find errors in the algorithm detection, operation delays in detection, and the impact of both error and delay on the network's functionality and the reliability of the collected data. The comparison could also include resource usage and its impact on the network lifetime as well as on the application functionality.

9.7 Publications Based on Part of this Thesis

- Yaqoob J Al-raisi and David Parish," *Tracking Sensor Node Operation deviations in Wireless Sensor Networks*", Proceedings of the 2nd ACM workshop on Performance monitoring and measurement of heterogeneous, Chania, Crete Island, 2007 Greece pp.84-87.
- Yaqoob J Al-raisi and David Parish," *Wireless Sensor Networks Performance Monitoring*", International Conference on Sensor Technologies Applications, 2007, pp. 277-282,Valencia, Spain.
- Yaqoob J Al-raisi and David Parish," *Approximate Wireless Sensor Network Health Monitoring* ", international Wireless Communication and Mobile Computing Conference 2007, Honolulu, Hawaii, USA, 2007.
- Yaqoob J Al-raisi and David Parish," Wireless Sensor Networks Performance Monitoring", Next Generation Networking Workshop 2007 (NGN2007), Oxford, UK, 2007.
- Yaqoob J Al-raisi and David Parish," *Performance Measurements in Wireless Sensor Networks* " PgNet 2005, Liverpool, UK, 2005.

For Submission

- Yaqoob J Al-raisi and David Parish," *Voting Median Base Algorithm for Approximate Wireless Sensor Networks Performance Measurement*", to be appear in IEEE Telecommunication Transactions.
- Yaqoob J Al-raisi and David Parish," *Wireless Sensor Networks Performance Measurements and Monitoring*", to be appear in ACM Transactions Sensor

Reference List

[1] K. Holger and W. Andreas, *Protocols and Architectures for Wireless Sensor Networks.* New York: John Wiley and Sons Ltd., 2005, pp. 524.

[2] I. Akyildiz F., S. Weilian, Y. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks," Communications Magazine, IEEE, vol. 40, pp. 102-114, Aug. 2002.

[3] K. Romer and F. Mattern, "*The Design Space of Wireless Sensor Networks,*" Wireless Communications, IEEE, vol. 11, pp. 54-61, Dec. 2004.

[4] R. Praveen, M. Ravi, G. Shashidhar and S. Udit, "*Survey of Sensor Networks,*" University of Texas, Dallas, Tech. Rep. UTDCS-10-03, July, 2001.

[5] K. Holger and W. Andreas, *Wireless Sensor Networks: Architectures and Protocols.* , vol. 3, Florida, USA: CRC Press, 2003, pp. 1-113.

[6] K. Mauri, H. Marko and D. Timo Hämäläinen, "*A Survey of Application Distribution in Wireless Sensor Networks,*" EURASIP Journal on Wireless Communications and Networking, vol. 2005, pp. 774-788, Dec. 2005.

[7] F. Koushanfar, M. Potkonjak and A. Sangiovanni-Vincentelli, "On-line Fault Detection of Sensor Measurements," in Sensors, 2003. Proceedings of IEEE, 2003, pp. 974-979.

[8] E. Eiman and N. Badri, "Cleaning and Querying Noisy Sensors," in the First ACM Conference on Embedded Networked Sensor Systems (SenSys'03), 2003, pp. 78-87.

[9] O. Akan B. and I. Akyildiz F., " ESRT: *Event-to-sink Reliable Transport in Wireless Sensor Networks,*" Networking, IEEE/ACM Transactions on, vol. 13, pp. 1003-1016, Oct. 2005.

[10] W. Yuan, S. Krishnamurthy V. and S. Tripathi K, "Improving the Reliability of Event Reports in Wireless Sensor Networks," in Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2004, pp. 220-225.

[11] Zhoa Jerry and G. Ramesh, "Understanding Packet Delivery Performance in Dense Wireless Sensor Networks," in Proceedings of the ACM SenSys Conference, 2003, pp. 1-13.

[12] Y. Weizhong and G. Kai, "Sensor Validation and Fusion for Gas Turbine Vibration Monitoring," System Diagnosis and Prognosis: Security and Condition Monitoring, vol. 5107, pp. 106-117, Aug. 2003.

[13] Wen Yao-jung, "Smart Dust Sensor Mote Characterization, Validation, Fusion and Actuation," Master Thesis, University of California Berkeley, 2004.

[14] Z. Yonggang, "Measurement and Monitoring in Wireless Sensor Networks," PhD Thesis, Computer Science Department, University of Southern California, USA, June. 2004.

[15] Multihop Routing, March 2007, available: http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html.

[16] I. Demirkol, C. Ersoy and F. Alagoz, "*MAC Protocols for Wireless Sensor Networks: a Survey* " IEEE Communications Magazine, vol. 44, pp. 115-121, April. 2006.

[17] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, David Culler," TinyOS: An Operating System for Sensor Networks", Rabaey J (Ed): 'Ambient Intelligence', Springer, 2004.

[18] Philip Levis, February 2006,"TinyOS Programming", March 2007, Available: http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf

[19] Sameer Tilak, Nael B. Abu-Ghazalah and Wendi Heizelman, "*Infrastructure Tradeoffs for Sensor Network*," in Workshop on Sensor Networks and Applications (WSNA'02), Atlanta, GA, 2002.

[20] D. Chen and P. Varshney K., "*QoS Support in Wireless Sensor Networks: A Survey,*" in 2004 International Conference on Wireless Networks (ICWN 2004), 2004, pp. 227-233.

[21] C. M. Vuran, B. O. Akan and F. I. Akyildiz, "*Spatio-Temporal Correlation: Theory and Applications for Wireless Sensor Networks,*" Computer Networks Journal (Elsevier), vol. 45, pp. 245-261, June. 2004.

[22] N. Jamal Al-Karaki and E. Ahmed Kamal, "*Routing Techniques in Wireless Sensor Networks: A Survey,*" Wireless Communication, IEEE, vol. 11, pp. 6-28, Dec. 2004.

[23] Bhaskar Krishnamachari, S. Sitharama Iyengar, "Efficient and Fault-Tolerant Feature Extraction in Wireless Sensor Networks", Information

Processing in Sensor Networks (IPSN 2003), Palo Alto, CA, USA, pp. 488-501, 2003.

[24] D. Amol, G. Carlos and R. Samuel Madden, "Model-*driven Data Acquisition in Sensor Networks*," in International Conference on very Large Databases (VLDB 2004), 2004, pp. 588-599.

[25] Y. Yan, "*Scalable, Synthetic, Sensor Network Data Generation*"," PhD Thesis, Computer Science Department, University of California, Los Angeles, USA, Jan. 2005.

[26] Laura K. Balzano," Addressing Fault and Calibration in Wireless Sensor Networks", Master Thesis, University of California, Los Angeles, 2007.

[27] Shoubhik Mukhopadhyay, Debashis Panigrahi and Sujit Dey, "*Model Based Error Correction for Wireless Sensor Networks*," in First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON '04), 2004, pp. 575-584.

[28] Prasad V. and Son S. H.," Classification of Analysis Techniques for Wireless Sensor Networks", The Fourth International Conference on Networking Sensing Systems(INSS '07),2007, pp.93-97

[29] Qing Cao, Ting Yan, John Stankovic, and Tarek Abdelzaher, " Analysis of Target Detection Performance for Wireless Sensor Networks," In *International Conference on Distributed Computing in Sensor Networks (DCOSS 2005)*, pages 84–89. TeX Users Group, June 2005.

[30] Ting Yan,". Analysis Approaches for Predicting Performance of Wireless Sensor Networks," PhD thesis, University of Virginia, August 2006.

[31] Chi-Fu Huang and Yu-Chee Tseng," The coverage problem in a wireless sensor network," In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications (WSNA)*, pages 115–121,2003.

[32] Enrique J., Duarte-Melo, and Mingyan Liu," Data-gathering wireless sensor networks: Organization and capacity", *Computer Networks(COMNET): Special Issue on Wireless Sensor Networks*, 43:519–537, November 2003.

[33] Liudong Xing, and Akhilesh Shrestha," QoS reliability of hierarchical clustered wireless sensor networks," In *Proceedings of the 25th IEEE International Performance, Computing, and Communications Conference,(IPCCC 2006)*, April 2006.

[34] Xiaorui Wang, Guoliang Xing, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill," Integrated coverage and connectivity

configuration in wireless sensor networks," In *Proceedings of ACM SenSys*, November 2003.

[35] Laurent Eschenauer and Virgil D. Gligor," A key-management scheme for distributed sensor networks," In *Proceedings of ACM Computer and Communications Security (CCS)*, 2002.

[36] Enrique J. Duarte-Melo and Mingyan Liu," Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks," In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, pages 21–25. IEEE Communications Society, November 2002.

[37] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala ." Improving Simulation for Network Research", Technical Report 99-702b, University of Southern California, September 1999.

[38] Elias Ekonomou, and Kate Booth," Simulating Sensor Networks", the 6th Annual PostGraduate Symposium on the convergence of Telecommunications, Networks, and Broadcasting, Liverpool John Moores University, UK, 27-28 June 2005.

[39] F. Ian Akyildiz, C. Mohmet Vuran and B. Ozgur Akan, "On Exploiting Spatial and Temporal Correlation in Wireless Sensor Networks," in Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (Wiopt'04), 2004, pp. 71-80.

[40] A. Raquel Mini, A. Antonio Loureiro and N. Badri, "Prediction-based Energy Map for Wireless Sensor Networks," in Simpsio Brasilerio De Redes De Computadores, 2003, pp. 165-169.

[41] H. Song and C. Edward, "*Continuous Residual Energy Monitoring in Wireless Sensor Networks*," in International Symposium on Parallel and Distributed Processing and Applications (ISPA 2004), 2004, pp. 169-177.

[42] NS2, January 2008, available: http://www.isi.edu/nsnam/ns/

[43] NRL, January 2008, available: http://cs.itd.nrl.navy.mil/work/proteantools/ns2extensions.php

[44] OMNeT++,January 2008, available: http://www.omnetpp.org/

[45] GLOMOSiM, January 2008, available: http://pcl.cs.ucla.edu/projects/glomosim/

[46] SENSE, January 2008, available: http://www.ita.cs.rpi.edu/sense/index.html

[47] Philip Levis and Nelson Lee,"*TOSSIM: a Simulation for TinyOS Networks*", Version 1.0, March 2007, http://www.tinyos.net/tinyos-1.x/doc/nido.pdf.

[48] OPNET, Janyary 2008, http://www.dei.unipd.it/wdyn/?IDsezione=2012

[49] Duane Hanselman and Bruce Littlield,"MAstring MATLAB 6",Prentice Hall, Upper Saddle River, New Jersy, USA, 2001.

[50] David Hand, Heikki Mannila, Padhraic Smyth, "*Data Mining*" The MIT Press Cambridge, England,2001,pp546.

[51] John K. Taylor and Cheryl Cihon, *Statistical Techniques for Data Analysis.*, Second edition .Boca Raton London: Chapman & Hall/CRC, 2004.

[52] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. *Introduction to Algorithms*, second edition, MIT Press and McGraw-Hill, 2004, pp 5-90.

[53] Unter der Leitung von," *Sensor Fusion in Time-Triggered Systems*", PhD thesis, Institut f¨ur Technische Informatik 182, eingereicht an der Technischen Universit¨at Wien, Fakult¨at f¨ur Technische Naturwissenschaften und Informatik, 2002.

[54] Siliva Santini, "Towards adaptive wireless sensor networks," Adjunct proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN 2006), Zurich, Switzerland, February 2006.

[55] Caimu Tang and Cauligi S. Raghavendra, "Correlation Analysis and Applications in Wireless Microsensor Networks," in *Mobile and Ubiquitous Systems: Networking and Services (MOBIQUITOUS 2004)*, 2004, pp. 184-193.

[56] G. C. Guiling Wang, Wensheng Zhang and T. La., "On Supporting Distributed Collaboration in Sensor Networks," *In Proceedings of the IEEE Military Communications Conference (MILCOM)*, 2003, pp. 1-8.

[57] Intel Lab "Intel Lab Experiment Data Set," March 2007, http://berkeley.intel-research.net/labdata/.

[58] UC Berkeley University, "Sonoma Redwood Sensor Network Deployment data set," March 2007, http://www.cs.berkeley.edu/~get/sonoma/.

[59] S. Robotti, Network Management Strategies, 1st edition, Computer Technology Research Corporation, Holtsville, NY, 1990, pp. 1-93.

[60] W. L. Lee, A. Datta, and R. Cardell-Oliver, Network Management in Wireless Sensor Networks, to appear in Handbook on Mobile Ad Hoc and Pervasive Communications, edited by M. K. Denko and L. T. Yang, American Scientific Publishers, 2006.

[61] Hsin Chih-fan and L. Mingyan, "Self-monitoring of Wireless Sensor Networks," Computer Communications, vol. 29, pp. 462-476, 2006.

[62] N. Ramanathan, L. Balzano, M. Burt, D. Estrin, T. Harmon, C. Harvey, J. Jay, E. Kohler, S. Rothenberg and M. Srivastava, "*Rapid Deployment With Confidence: Calibration and Fault Detection in Environmental Sensor Networks*," Centre for Embedded Networked Sensing (CENS), Tech. Rep. #62, July, 2006.

[63] C. Jaikaeo, C. Srisathapornphat and C. Shen, "Diagnosis of Sensor Networks," in Communications, 2001. ICC 2001. IEEE International Conference, 2001, pp. 1627-1632.

[64] Nasir Mehranbod," A Probabilistic Approach for Sensor Fault Detection and Identification," Doctor Thesis, Drexel University, November 2002.

[65] W. Yao-jung , M. Alice Agogine and G. Kai, "*Fuzzy Validation and Fusion for Wireless Sensor Networks*," in ASMEinternational Mechanical Engineering Congress and RD&D Expo (IMECE2004),Anaheim, California, *USA*, 2004.

[66] A. Mohamed, K. Parameshwaran and F. Jeff, "*A methodology for the Fusion of Redundant Sensors*," in American Control Conference 2000, pp. 2922-2926.

[67] L. Chang-Tien, C. Dechang and Z. Yufeng, "*Algorithms for Spatial Outlier Detection*," in Third IEEE International Conference on Data Mining(ICDM'03), 2003, pp. 597-600.

[68] M. Ding, D. Chen, K. Xing and X. Cheng, "Localized Fault-tolerant Event Boundary Detection in Sensor Networks," in IEEE INFOCOM 2005, 2005, pp. 902-913.

[69] G. Saurabh, K. Aman and B. Mani Srivastava, "*Self Aware Actuation for Fault Repair in Sensor Networks,*" in IEEE International Conference on Robotics and Automation, 2004, pp. 5244-5249.

[70] G. Indranil, V. Robbert Renesse and P. Kenneth Birman, "*Scalable Fault-tolerant Aggregation in Large Process Groups,*" in *The 2001* International Conference on Dependable Systems and Networks, 2001, pp. 433-442.

[71] C. T., S. K. and R. P., "*Fault Tolerance in Collaborative Sensor Networks for Target Detection,*" IEEE Transactions on Computers, vol. 53, pp. 320-333, March 2004. 2004.

[72] F. Koushanfar, M. Potkonjak, A. Sangiovanni-Vincentelli,"*Fault Tolerance Techniques for Wireless Ad Hoc Sensor Networks*," IEEE Sensors, vol. 2, pp. 1491-1496, June. 2002.

[73] X. Luo, M. Dong and Y. Huang, "*On Distributed Fault-tolerant Detection in Wireless Sensor Networks*," IEEE Transactions on Computers, vol. 55, pp. 58-70, June. 2006.

[74] J. Cheryan and J. Ranjit Mathai, "*Fault Tolerance in Sensor Networks: A Survey of Fault Tolerant Sensor Network Algorithms and Techniques*," University of Wisconsin-Madison, USA, Tech. Rep. ECE 753, Spring, 2004.

[75] Linnyer Beatrys Ruiz, Isabela G. Siqueria and Leonardo B. Oliveira, "*Fault Management in Event-driven Wireless Sensor Networks*," in MSWiM'04, October 4-6, Venezia, Italy, 2004.

[76] Jinran Chen, Shubha Kher and Arun Somani, "*Distributed Fault Detection of Wireless Sensor Networks*," in the 2006 Workshop on Dependability Issues in Wireless Ad Hoc Networks and Sensor Networks Table of Contents 2006, pp. 65-72.

[77] Nithya Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler and D. Estrin, "Sympathy for the Sensor Network Debugger," in The 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys 2005), 2005, pp. 255-267.

[78] Stefano Chessa and Paolo Santi, "Crash Faults Identification in Wireless Sensor Networks," Computer Communications, vol. 25, pp. 1273-1282, Sept. 2002.

[79] Linnyer B. Ruiz, José Marcos S. Nogueira and Antonio A. F. Loureiro, "Manna: A management architecture for wireless sensor networks," IEEE Communications Magazine, vol. 41, pp. 116-125, February. 2003.

[80] T. Bokareva, S. Jha and N. Bulusu, "SASHA: Toward a Self-healing Hybrid Sensor Network Architecture," in the 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II), IEEE, 2005, pp. 71-78.

[81] J. Steyer, L. Lardon and O. Bernard, "*Sensors Network Diagnosis in Anaerobic Digestion Processes Using Evidence Theory*," Water Science and Technology, vol. 50, pp. 21-29, 2004.

[82] C. Shen, C. Srisathapornphat and C. Jaikaeo, "*Sensor Information Networking Architecture and Applications*," IEEE Personnel Communication Magazine, vol. 8, pp. 52-59, Aug. 2001.

[83] Guillermo Heredia Benot, Anibal Ollero Baturone, Rajesh Ishwar Mahtani Mahtani, Manuel Béjar Domínguez, Volker Remuss and Marek Musial, "*Detection of Sensor Faults in Autonomous Helicopters*," in Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference, 2005, pp. 2229-2234.

[84] G. Anastasi, A. Falchi, A. Passarella, M. Conti and E. Gregori, "*Performance Measurements of Motes Sensor Networks*," in MSWiM '04: Proceedings of the 7Th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2004, pp. 174-181.

[85] B. Phil, G. David, M. Joseph Hellerstein, H. Wei and M. Samuel, "*TASK: Sensor Network in a Box*," in Second IEEE European Workshop on Wireless Sensor Networks and Applications (EWSN), 2005, pp. 133-144.

[86] G. Carlos, B. Peter, T. Romain, P. Mark and M. Samuel, "*Distributed Regression: An Efficient Framework for Modeling Sensor Network Data*," in Information Processing in Sensor Networks (IPSN 2004), Berkeley, 2004.

[87] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "*An Aanalysis of a Large Scale Habitat Monitoring Application*," in the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04), 2004, pp. 214-226.

[88] Crossbow Technology, "*Getting started guide*," Manual No. PN: 7430-0022-07, September, 2005.

[89] Crossbow Technology, "*Mote-view 1.2 user's manual*," Manual No. PN: 7430-0008-03, October, 2006.

[90] Millennial Net, "EK-5209-5 evaluation kit, i-bean 916 MHz wireless sensor network User's guide," Document No. DOC-0005, June, 2004.

[91] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho and M. A. Perillo, "*Middleware to Support Sensor Network Applications*," IEEE Network, vol. 18, pp. 6-14, 2004.

[92] S. Rhee, D. Seetharam and S. Liu, "Techniques for Minimizing Power Consumption in Low Data-rate wireless Sensor Networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2004, pp. 1727-1731.

[93] Sapon Tanachaiwiwat and Ahmed Helmy, "Correlation Analysis for Alleviating Effects of Inserted Data in Wireless Sensor Networks," in *Mobile and Ubiquitous Systems: Networking and Services, 2005 (MobiQuitous 2005)*, 2005, pp. 97-108.

[94] Ricardo Gutierrez-Osuna, Wright state university, "Sensor Characteristics," March 2007, http://courses.cs.tamu.edu/rgutier/ceg499_s02/l2.pdf.

[95] A. Agogino, K. Goebel and S. Alag, "Intelligent Sensor Validation and Sensor Fusion for Reliability and Safety Enhancement in Vehicle Control," California PATH Research, Tech. Rep. UCB-ITS-PRR-95-40, 1995.

[96] Jari Nasi and Aki Sorsa, "On-line Measurement Validation through Confidence Level Based Optimal Estimation of a Process Variable," OULU University control engineering labortry, Tech. Rep. 25, December, 2004.

[97] Satnam Alag, Alice M. Agogino and Mahesh Morjaria, "A Methodology for Intelligent Sensor Measurement, Validation, Fusion, and Fault Detection for Equipment Monitoring and Diagnostics," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* vol. 15, pp. 307-320, Sep. 2001.

[98] Ossama Ypunis and Sonia Fahmy, "An Experimental Study of Routing and Data Aggregation in Sensor Networks," in *IEEE International Mobile Ad hoc and Sensor Systems Conference,* 2005, pp. 8-14.

[99] Vladimir Bychkovskiy, Seapahn Megerian, Deborah Estrin, and Miodrag Potkonjak., "A Collaborative Approach to In-Place Sensor Calibration", In Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03), pp 301-316. Springer-Verlag Inc., 2003.

[100] J. Staddon, D. Balfanz and G. Durfee, "*Efficient Tracing of Failed Node in Sensor Networks,*" in WSNA 02: Proceedings of the 1St ACM International Workshop on Wireless Sensor Networks and Applications, 2002, pp. 122-130.

[101] Duane L. Mattern, Link C Jaw, Ten-Huei Guo, Ronald Graham and William McCoy, "Using Neural Networks for Sensor Validation," NASA, USA, Tech. Rep. WU-519-30-53-00, July, 1998.

[102] Themistoklis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki and Dimitrios Gunopulos, "Distributed Deviation Detection in Sensor Networks," *ACM SIGMOD,* vol. 32, pp. 77-82, December. 2003.

[103] J. Branch, B. Szymanski, C. Giannella, Ran Wolff and H. Kargupta, "In-network Outlier Detection in Wireless Sensor Networks," in *IEEE International Conference on Distributed Computing Systems 2006 (ICDCS 2006),* 2006, pp. 51-57.

[104] Xiuli Ma, Dongqing Yang, Shiwei Tang, Qiong Luo, Dehui Zhang and Shuangfeng Li, "Online mining in sensor networks," in *Network and Parallel Computing 2004 (NPC 2004),* 2004, pp. 544-550.

[105] Larkey L. B., Bettencourt M. A., and Hagberg A. A., (2006),"In-Situ Data Quality Assurance for Environmental Applications of Wireless Sensor Networks," Los Alamos National Laboratory Unclassified Report LA-UR-06-1117, available:

http://homepage.psy.utexas.edu/homepage/students/Larkey/Larkey-Bettencourt-Hagberg-2006.pdf.

[106] J. Frolik, M. Abdelrahman and P. Kandasamy, "A Confidence-based Approach to the Self-validation, Fusion and Reconstruction of Quasi-redundant Sensor Data," *Instrumentation and Measurement, IEEE Transactions,* vol. 50, pp. 1761-1769, Dec. 2001.

[107] S.R. Jeffery, G. Alonso, M.J. Franklin, W. Hong and J. Widom, "Declarative Support for Sensor Data Cleaning," in *Fourth International Conference on Pervasive Computing,* 2006, pp. 83-100.

[108] David J. Olive, "A Simple Confidence Interval for the Median," National Science Foundation, Tech. Rep. DMS 0202922, June 3, 2005.

[109] David Wagner, "Resilient Aggregation in Sensor Networks", in SASNA'04: Proceedings of the Security of ad hoc and Sensor Networks. ACM press, 2004, pp 78-87.

[110] Mengxia Zhu, Song Ding, Richard R. Brooks, Qishi Wu, Nageswara S.V. Rao, and S. Sitharama Iyengar "Fusion of Threshold Rules for Target Detection in Sensor Networks", ACM Transactions on Sensors Networks, 2005.

[111] Jason L. Hill, "System Architecture for Wireless Sensor Networks," PhD Thesis, Computer Science Department, University of California, Berkeley, USA, spring 2003.

[112] U.S. Environmental Protection Agency "CASTNet 2002 Quality Assurance Report", March 2007, http://www.epa.gov/castnet/library/qanualoz.html.

[113] F. Araujo and L. Rodrigues, "On the Monitoring Period for Fault-tolerant Sensor Network," the second Latin-American Symposium on Dependable Computing, Salvador, Bahia, Brazil, October 2005.

[114] Y. Sankarasubramaniam, I.F. Akyildiz and S. W. McLaughlin, "Energy Efficiency based Packet Size Optimization in Wireless Sensor Networks," in *1St IEEE Intl. Workshop on Sensor Network Protocols and Applications (SNPA),* 2003, pp. 1-8.

[115] M. Hempstead, V. Shnayder, and B. rong Chen," *Power TOSSIM: Efficient power simulation for TinyOS applications*", Report No. CS263, March 2007, http://www.eecs.harvard.edu/~shnayder/ptossim/.

[116] Crossbow Company, "Crossbow 2006 Wireless Sensor Networks Product Reference Guide", March 2007 http://www.xbow.com/Order/downloadcatalogrequest.aspx.

[117] MicroStrain Company, March 2007, Available: http://www.microstrain.com.

[118] Ember Company, March 2007, Available: http://www.ember.com/ .

[119] Millennial Net Company, March 2007, Available: http://www.millennial.net.

[120] David Gay, Phil Levis, Rob von Behren, MattWelsh, Eric Brewer, and David Culler," The nesC language: A holistic approach to networked embedded systems", In Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003.

[121] TinyOs Documentation, March 2007, Available: http://www.tinyos.net/tinyos-1.x/index.html.

[122] Jeff Thorn, March 2005 "Deciphering TinyOS Serial packets" Octave Tech. Brief #5-01, avalible:http://www.octavetech.com/pubs/TB5-01%20Deciphering%20TinyOS%20Serial%20Packets.pdf

[123] Valliappan Annamalai," Introduction to Programming Mica2 Motes Using TinyOS, "March 2007, Available: http://www.shamir.eas.asu.edu/~mcn/cse494fa05/tinyos-introduction.doc.

[124] R.J. Potten and J. Chen, "Observer-based Fault Detection and Isolation: Robustness and Applications," Control Engineering Practice, Vol. 5, No. 5, May, 1997, pp. 671-682.

[125] Robert Szewczk, Joseph Polastre, Alan Mainwaring and David Culler, "Lessons from a Sensor Network Expedition," in *1St European Workshop on Wireless Sensor Networks (EWSN 04),* 2004, pp. 307-322.

[126] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "*A Macroscope in the Redwoods*", ACM Conference on Embedded Networked Sensor Systems (SenSys'05),2005, pp. 51-63.

Appendix A – Mote2 Power Consumption Model.

Processor	Ipa=8 mA(1%)	Ips=15 μA(99%)	
Sensor	Isa=5 mA(5%)	Iss=5 μA(95%)	
Flash memory	Imr=4 mA(0%)	Imw=15 mA(0%)	Ims=2 μA(100%)
Radio	Itx=16.8 mA	Irx=10 mA(5%)	Ics=1 μA(95%)
0 dBm(1mW) transmission power 62.4μsec time to transmit one bit 1 minute reporting rate			
Battery	Total power 2.2 A-hr, voltage= 3 v	Total energy available Emax=Vb*Ab*3.6*1000 =2.2*3*3600=23760J	Battery
Energy consumption in reporting time E0=P0*t0 P0=Vb*Ieff= Vb*(IpaRpa+ IpsRps+ IsaRsa + IssRss + ImrRme + ImwRmw + ImsRms) E0=3*(8*0.05+0.015*0.95+5*0.05+0.005*0.95+ 4*0 + 15*0 +0.002*1) *60=63mJ For 34 byte packet, packet transmission time tp=0.0624*K*8=0.0624*34*8=16.97msec Etx=Ptx*tp, Ptx=Vb*Itx = 3*16.8*16.97=0.86mJ per packet Erx=Prx*trx , Prx=Vb*Irx =3*0.1*0.05*60 =90 mJ per packet Ecs(radio sleep)= 0.000001*3*0.95*60=171μJ Energy leakage in application Motes of around 0.1 mA			
Total Energy: Etotal=Eerg+Etx+Erx+Ecs =63+0.86+90+0.171=154 mJ Total life time = Ebatt/(Etotal*60*24)=154285.7/(60*24)=107.1 days=3.57 months			

Table A-1. Power Consumption Model

Ipa, Ips, Isa current drawn by the processor during the active and sleeping period respectively.
Issa, Iss current drawn by the sensor during the active and sleeping period respectively.
Imr , Imw, Imw current drawn for memory read, write and sleep respectively.
Itx, Irx, Ics current drawn be radio transmit, receive and sleep modes respectively.

Appendix B- Packet Losses in Intel Lab Data Set

Node	Dead before node 1	Total lost packets	Percentage of loss	Network loss	Network percentage loss	Synch. Loss	Synch. Loss percentage	Synch./loss	Net./loss
2	---	64174	72	44987	50	19187	21.5	30	70
3	88841	62652	70	42539	48	20113	23	32	68
4	86278	64740	72	45372	51	19368	22	30	70
29	---	53908	60	32514	37	21394	24	40	60
31	---	50139	56	25556	29	24583	28	49	51
32	85515	63425	71	46028	52	17397	18	27	73
33	---	70305	79	55504	62	14801	17	21	79
34	---	63536	71	50525	56	13011	15	21	79
35	75941	59911	67	37878	42	22033	25	37	63
36	---	58547	66	36927	41	21620	24	37	63
37	---	64123	72	46145	52	17978	20	28	72
39	---	73270	82	60719	68	12551	14	17	83

Table B.1. Node 1 Neighbors with 89097 Events

Appendix C - Some of Performance Measurements Tools in Wired Networks.

Type	Used	Place
Simple network management protocol (SNMP)	Used to gather statistics from routers and switches including number, size of IP packets, total bytes, router CPU utilization, and discarded packets	WAN/LAN/broadband
Multirouter traffic grapher (MRTG)	Used to display bandwidth usage and other information over time	
OCXmon	Passive monitoring used to examine network traffic without introducing any traffic of their own	
NLANR Active Measurement Program (AMP)	Test bandwidth and delay between participating institutions (a ping test measuring routing trip time delay and trace route to show what networks are used between institutions)	
Surveyor	One way delays are measured between Surveyor machines at participating institution (asymmetries in the network are revealed that normal round-trip time delays do not)	
Treno "TCP Reno"	Emulates the TCP protocol stack using UDP. It is used to compare an operating system's TCP implementation with a modem TCP implementation that includes such improvements as SACK (selective acknowledgement), FACK (forward acknowledgement), and MTU Discovery. It allows targeting individual routes along the path, to discover what links are problematic	End to End
mping	Stress the network intentionally flooding the route queues to test queuing properties (find bandwidth and packet losses as TCP window size increases)	
IPerf	Measures the maximum TCP bandwidth and the UDP performance between two machines, packet losses and variation in delay)jitter)	
Traceroute	Used to fined the path your data takes through the network which provide round-trip time measurements to each route	
Ping	Repetitively find round-trip time measurements to a particular machine or route.	
Matts' Trace Route (mtr)	Combine functionality of traceroute and ping and presents the output data in an easy to read tabular format (repetitively pings each router along the path, showing delay and packet loss)	
tcpdump	Standard UNIX utility to examine or "sniff" the traffic on the network	
tcptrace	Can be used to analyze the output from tcpdump and xplot will show the packets graphically	
Xplot	Helping to reveal "pathological " network behavior	
Netlog	It logs periodic bandwidth results, giving a fine-grained picture of how an application interacts with the network	Application
Viznet	Visualizes the raw performance data that Vetlog provides	
Real-time transport protocol (RTP)	Detect delay, jitter and packet loss	

Appendix D – VMBA Flowchart.

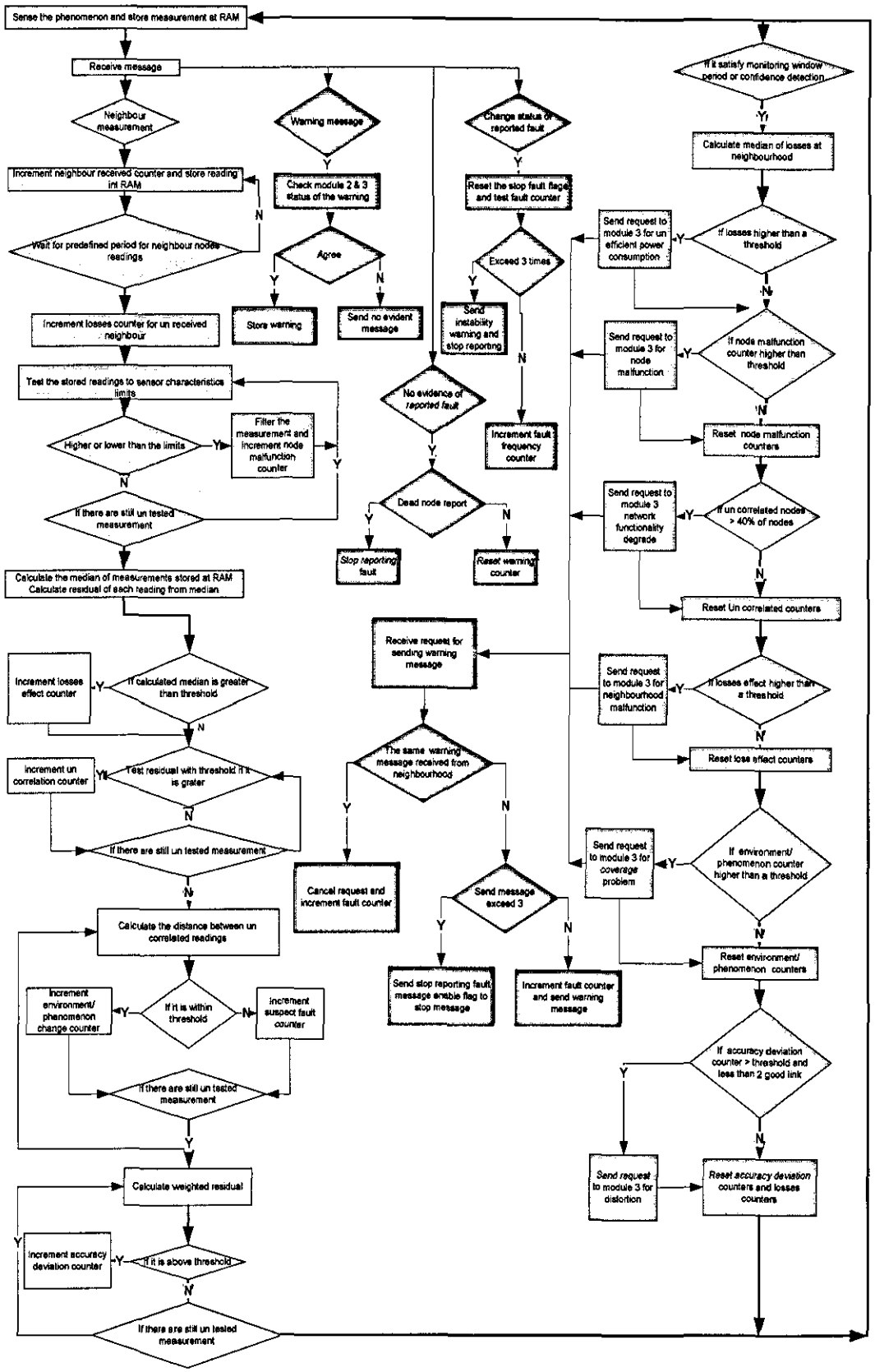

Figure D-1. VMBA Algorithm Flowchart

Appendix E- VMBA Algorithm Resources use Estimation Tables

	Pseudo code	Cost	Time	Memory	Tx	Rx
1	Δ create a random event number for monitor	0	0	0	0	0
2	do key1_1[i] ← Random number or key1_6	C1	1	1	0	0
3	Δ check if the event number of monitor is the same	0	0	0	0	0
4	while interval= key1_1[i]	C2	1	1	0	0
5	for j ← 1 to length[A]	C3	n	n	0	0
6	do key1[j] ← loss A[j]	C4	n-1	n-1	0	0
7	Do key2[event] ←median[A]	C5	1	1	0	0
8	Do key3[j] ← residual A[j]	C6	n	n	0	0
9	Do key4[j] ← weight A[j]	C7	n	n	0	0
10	while key3[j] > correlated margin	C8	0	0	0	0
11	do key5[j] ← 1	C9	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
12	Δ calculating losses	0	0	0	0	0
13	while key3[j]> threshold_divert and	0	0	0	0	0
14	key4[j] <threshold_weight and key5[j]=1	C10	1	1	0	0
15	do key6[j] ← 1	C11	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
16	while window > window_loss1	C12	1	1	0	0
17	while key1[j] > threshold _loss1	C13	1	1	0	0
18	do key7[j] ← 1	C14	$\sum_{j=1}^{n} j$	1	0	0
19	do key1[j] → send	C15	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0
20	do key1[j] ← receive	C16	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	$\sum_{j=1}^{n} j$
21	do key1[A] ← reset	C17	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
22	while window > window_loss2	C18	1	1	0	0
23	while key7[j] >threshold_loss3	C19	1	1	0	0
24	do key7[j] → send	C20	$\sum_{j=1}^{n} j$	1	$\sum_{j=1}^{n} j$	0
25	do key7[j] ← receive	C21	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	$\sum_{j=1}^{n} j$
26	do key6[A] ← reset	C22	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
27	Δ calculating distortion level	0	0	0	0	0
28	for j ← 1 to length[A]	C23	n-1	n-1	0	0
29	do key1_3[j] ← accuracy A[j]	C24	n-1	n-1	0	0
30	while key1_3[j] < accuracy_level	C25	1	1	0	0
31	do key1_4[j] ← 1	C26	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
32	while key1_4[j] >threshold_accuracy	C27	$\sum_{j=1}^{n} j$	1	0	0
33	do key1_4[j] → send	C28	$\sum_{j=1}^{n} j$	1	$\sum_{j=1}^{n} j$	0
34	do key1_5[j] ← receive	C29	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	$\sum_{j=1}^{n} j$
35	do key1_4[A] ← reset	C30	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
36	Δ calculating reading diversion	0	0	0	0	0
37	while window > small_window	C31	1	1	0	0
38	while key6[j] > threshold _ small_window	C32	$\sum_{j=1}^{n} j$	1	0	0
39	do key8[j] ← 1	C33	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
40	do key6[j] → send	C34	$\sum_{j=1}^{n} j$	1	$\sum_{j=1}^{n} j$	0
41	do key6[j] ← receive	C35	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	$\sum_{j=1}^{n} j$
42	do key6[A] ← reset	C36	$\sum_{j=1}^{n} j$	0	0	0
43	while window > big _window	C37	1	1	0	0
44	while key8[j] >threshold_ window_ big	C38	1	1	0	0
45	do key8[j] → send	C39	1→n	1	$\sum_{j=1}^{n} j$	0

	Pseudo code	Cost	Time	Memory	Tx	Rx
46	do key8[j] ← receive	C40	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	$\sum_{j=1}^{n} j$
47	do key8[A] ← reset	C41	$\sum_{j=1}^{n} j$	0	0	0
48	Δ calculating next window monitoring time	0	0	0	0	0
49	while key1_5[j] > threshold _ accuracy2	C42	1	1	0	0
50	do key1_6[j] ← interval+1	C43	$\sum_{j=1}^{n} j$	1	0	0
51	while key1_5[j] < threshold _ accuracy2	C44	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0
52	do key1_6[j] ← interval+accuracy*small_window	C45	$\sum_{j=1}^{n} j$	0	0	0
53	Δ removing node	0	0	0	0	0
54	remove A[j]	C46	$\sum_{j=1}^{n} j$	$\sum_{j=1}^{n} j$	0	0

Table E-1. Pseudo Code for the Proposed Algorithm

n is the total number of nodes appearing in the algorithm time interval calculation due to losses.

Algorithm Message		Max. No. of packets transmitted at node level	Max. No. of packet received at node level
Node malfunction	Detect	4n	$4n^2$
	No evident	4n	$4n^2$
	Total	8n	$8n^2$
Node dead	Detect	4n	4(n-dead)(n-dead)
	No evident	4(n-dead)	4(n-dead)(n-dead)
	Total	8n-4(dead node)	$8n^2 - 12n(dead\ node) + 4(dead\ node)^2$
Neighborhood malfunction	Detect	4	4n
	No evident	4	4n
	Total	8	8n
Neighborhood collected data accuracy	Detect	4n	$4n^2$
	No evident	4n	$4n^2$
	Total	8n	$8n^2$
Coverage (distortion)	Detect	4n	$4n^2$
	No evident	4n	$4n^2$
	Total	8n	$8n^2$
Connectivity	Detect	4	4n
	No evident	4	4n
	Total	8	8n
Max. Permanent detection packets		32n+16-4(dead nodes)	$32n^2 + 16n - 12n(dead\ nodes) + 4(dead\ nodes)^2$
Max. temporary detection packets		96n+36-12(dead nodes)	$98n^2 + 82n - 48n(dead\ nodes) + 16(dead\ nodes)^2$

Table E-2. Number of Transmitted Messages in both Permanent ant Temporarily Faults

Confidence value	Interval from uniformly distributed error	Statistical Variance
0	[-100.0,100.0]	3333.33
1	[-70.2,70.2]	1644.65
2	[-49.3,49.3]	811.47
3	[-34.7,34.7]	400.37
4	[-24.3,24.3]	197.54
5	[-17.1,17.1]	97.47
6	[-12.0,12.0]	48.09
7	[-8.4,8.4]	23.73
8	[-5.9,5.9]	11.71
9	[-4.2,4.2]	5.78
10	[-2.9,2.9]	2.85
11	[-2.1,2.1]	1.41
12	[-1.4,1.4]	0.69
13	[-1.0,1.0]	0.34
14	[-0.7,0.7]	0.17
15	[-0.5,0.5]	0.08

Table E-3. Confidence Values of the Weighted Moving Average Fusion (WMAV Fusion) Algorithm Used in TTP/A Protocol for Temperature Measure

Appendix F- Crossbow Different Node Modules and their Specifications

Mote Type	Mica	Mica2	Mica2Dot	Mica3	Micaz	Imote2
Microprocessor						
Type	AT90LS835	Atmega 128	Atmega 128	Atmega 128	Atmega 128	PXA271 XScale
CPU clock(MHz)	4	7.38	4	4	4	13-416 Mhz
Memory (KB)	8	128	128	128	128	256/32,000
RAM (KB)	0.5	4	4	4	4	256
UARTs	1	2	2	2	2	3
SPI	1	1	1	1	1	2
I2C	Software	Hardware	Hardware	Hardware	Hardware	-----
Radio communication						
Radio Type	RFM TR1000	Chipcon CC100		Chipcon CC1020	Chipcon CC2420	15.4 (BT/802.11)
Frequency	433/916 MHz (single frequency)	916/433/315 MHz (Multiple channels)			2.4 GHz (multiple channels)	2.4 GHz (16 channels)
Radio speed (Kbps)	40	38.4		76	250	250 (720/11,000)
Modulation scheme	ASK (amplitude shift keyed)	FSK		GFSK	O-QPSK	----
Encoding	SECDED (software)	Manchester (Hardware)			n/a	
Power		2 AA 1.5 V Batters	Coin size 3V Lithium	-----	-----	3 x AAA Batters 3.2-4.5 V

Appendix G- Alternate Effect on Algorithm Detection

```
FF FF 0B 7D 06 02 00 03 00 01 03
FF FF 0B 7D 06 06 00 03 00 06 02
FF FF 0B 7D 06 07 00 03 00 06 02
FF FF 0B 7D 06 04 00 03 00 06 02
FF FF 0B 7D 06 05 00 03 00 06 02
00 00 11 7D 0C 08 00 08 00 3F 02 01 00 FD 02 00 00
00 00 11 7D 0C 05 00 05 00 41 02 01 00 8B 03 00 00
FF FF FA 7D 1C 07 00 07 00 40 02 01 00 00 06 00 00 FE 08 00 FE 02 00 FE 03 00 FE 01 00 FD 04 00 F8
00 00 11 7D 0C 03 00 03 00 4C 02 01 00 78 03 00 00
FF FF 0B 7D 06 03 00 08 00 01 02
00 00 11 7D 0C 06 00 06 00 41 02 01 00 62 03 00 00
00 00 11 7D 0C 01 00 01 00 43 02 01 00 73 03 00 00
FF FF FA 7D 1C 08 00 08 00 40 02 01 00 00 06 00 00 FE 05 00 FE 03 00 FE 06 00 FE 01 00 FE 02 00 FE
00 00 11 7D 0C 07 00 07 00 41 02 01 00 89 03 00 00
00 00 11 7D 0C 04 00 04 00 42 02 01 00 63 03 00 00
FF FF 0B 7D 06 04 00 08 00 01 02
00 00 11 7D 0C 02 00 02 00 52 02 01 00 94 03 00 00
00 00 11 7D 0C 08 00 08 00 41 02 01 00 FC 02 00 00
00 00 11 7D 0C 05 00 05 00 42 02 01 00 8C 03 00 00
FF FF 0B 7D 06 05 00 08 00 01 02
00 00 11 7D 0C 03 00 03 00 4D 02 01 00 78 03 00 00
FF FF 0B 7D 06 06 00 08 00 01 02
00 00 11 7D 0C 01 00 01 00 44 02 01 00 72 03 00 00
00 00 11 7D 0C 07 00 07 00 42 02 01 00 89 03 00 00
FF FF 0B 7D 06 07 00 08 00 01 02
00 00 11 7D 0C 04 00 04 00 43 02 01 00 63 03 00 00
00 00 11 7D 0C 02 00 02 00 53 02 01 00 95 03 00 00
00 00 11 7D 0C 08 00 08 00 42 02 01 00 FB 02 00 00
FF FF 0B 7D 06 08 00 08 00 01 03
```

Appendix H- Dynamic Threshold Set Values from Empirical Experiments

	2 nodes	3 nodes	4 nodes	5 nodes	6 nodes	7 nodes
Multiple of 1	0	0	0.05	6	0	1
Multiple of 2	0	1.15	21.35	1.41	2.1	6
Multiple of 3	0	0	1.5	1.6	0	0.5
Multiple of 4	0	0	2.3	0	0.816	0.5
Multiple of 5	0	0	5	4.45	9.5	3.2
Multiple of 6	0	10.39	0	5.3	2.5	3.1
Average standard deviation	0	1.9	5	3.1	2.4	

Table H.1. Standard Deviation of Neighbor Nodes Threshold Using First Approach

	2 nodes	3 nodes	4 nodes	5 nodes	6 nodes
Multiple of 1	0	0	0.5	0	0
Multiple of 2	0	1.15	0	0	0
Multiple of 3	0	0	0	1.34	0
Multiple of 4	0	0	2	1.8	0
Multiple of 5	0	0	2.5	0	0
Multiple of 6	0	0	0	2.8	0
Average standard deviation	0	0.2	0.8	1	0

Table H.2. Standard Deviation of Neighbor Nodes Threshold Using Second Approach

Appendix I- VMBA Algorithm Self Configuration Experiment Packets

FF FF FA 7D 1B 04 00 04 00 43 00 01 00 00 65 03 05 07 00 FC 00 00 F4 03 00 DC 09 00 DC 05 00 DC
FF FF FA 7D 1B 06 00 06 00 43 00 01 00 00 96 03 05 00 00 FE 07 00 FC 09 00 DC 0A 00 DC 05 00 DC
FF FF FA 7D 1B 07 00 07 00 42 00 01 00 00 6C 03 05 00 00 FC 09 00 DC 06 00 DC 03 00 DC 04 00 DC
FF FF FA 7D 1B 08 00 08 00 43 00 01 00 00 92 03 05 00 00 FE 07 00 FB 02 00 DC 05 00 DC 09 00 DC
FF FF FA 7D 1B 09 00 09 00 43 00 01 00 00 8A 03 05 00 00 FE 07 00 FC 05 00 DC 01 00 DC 04 00 DC
FF FF FA 7D 1B 00 00 00 00 42 00 00 7E 00 00 00 05 07 00 FC 05 00 DC 01 00 DC 03 00 DC 04 00 DC
FF FF FA 7D 1B 0A 00 0A 00 43 00 01 00 00 09 02 05 00 00 FE 07 00 FC 01 00 DC 03 00 DC 02 00 DC
FF FF FA 7D 1B 02 00 02 00 44 00 01 00 00 8B 03 05 00 00 FE 07 00 FC 03 00 DC 04 00 DC 06 00 DC
FF FF FA 7D 1B 05 00 05 00 44 00 01 00 00 98 03 05 00 00 FE 07 00 FC 06 00 DC 08 00 DC 09 00 DC
FF FF FA 7D 1B 01 00 01 00 44 00 01 00 00 71 03 05 00 00 FE 07 00 FC 06 00 DC 08 00 DC 03 00 DC
FF FF FA 7D 1B 04 00 04 00 44 00 01 00 00 64 03 05 07 00 FC 00 00 F4 03 00 DC 09 00 DC 05 00 DC
FF FF FA 7D 1B 06 00 06 00 44 00 01 00 00 95 03 05 00 00 FE 07 00 FC 09 00 DC 0A 00 DC 05 00 DC
FF FF FA 7D 1B 07 00 07 00 43 00 01 00 00 6C 03 05 00 00 FC 09 00 DC 06 00 DC 03 00 DC 04 00 DC
FF FF FA 7D 1B 08 00 08 00 44 00 01 00 00 92 03 05 00 00 FE 07 00 FB 02 00 DC 05 00 DC 09 00 DC
FF FF FA 7D 1B 09 00 09 00 44 00 01 00 00 8C 03 05 00 00 FE 07 00 FC 05 00 DC 01 00 DC 04 00 DC
FF FF FA 7D 1B 00 00 00 00 43 00 00 7E 00 00 00 05 07 00 FC 05 00 DC 01 00 DC 03 00 DC 04 00 DC
FF FF FA 7D 1B 0A 00 0A 00 44 00 01 00 00 0C 02 05 00 00 FE 07 00 FC 01 00 DC 03 00 DC 02 00 DC
00 00 0B 7D 0F 02 00 02 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 02 00 02 00 46 00 01 00 00 8E 03 05 00 00 FE 07 00 FC 03 00 DC 04 00 DC 05 00 DC
00 00 0B 7D 0F 05 00 05 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 05 00 05 00 46 00 01 00 00 9A 03 05 00 00 FE 07 00 FC 06 00 DC 08 00 DC 09 00 DC
00 00 0B 7D 0F 01 00 01 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 01 00 01 00 46 00 01 00 00 75 03 05 00 00 FE 07 00 FC 06 00 DC 08 00 DC 05 00 DC
00 00 0B 7D 0F 03 00 03 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 03 00 03 00 46 00 01 00 00 6B 03 05 07 00 FC 06 00 DC 08 00 DC 09 00 DC 01 00 DC
00 00 0B 7D 0F 04 00 04 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 04 00 04 00 46 00 01 00 00 63 03 05 07 00 FC 00 00 F4 03 00 DC 09 00 DC 05 00 DC
00 00 0B 7D 0F 06 00 06 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 06 00 06 00 46 00 01 00 00 95 03 05 00 00 FE 07 00 FC 09 00 DC 05 00 DC 01 00 DC
00 00 0B 7D 0F 07 00 07 00 44 00 01 0A 00 01 0A 00 01 01 00
FF FF FA 7D 1B 07 00 07 00 45 00 01 00 00 6E 03 05 00 00 FC 09 00 DC 06 00 DC 03 00 DC 04 00 DC
00 00 0B 7D 0F 08 00 08 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 08 00 08 00 46 00 01 00 00 91 03 05 00 00 FE 07 00 FB 05 00 DC 09 00 DC 03 00 DC
00 00 0B 7D 0F 09 00 09 00 45 00 01 0A 00 01 0A 00 01 02 00
FF FF FA 7D 1B 09 00 09 00 46 00 01 00 00 8B 03 05 00 00 FE 07 00 FC 05 00 DC 01 00 DC 04 00 DC
FF FF FA 7D 1B 00 00 00 00 44 00 00 7E 00 00 00 05 07 00 FC 05 00 DC 01 00 DC 03 00 DC 04 00 DC
00 00 0B 7D 0F 0A 00 0A 00 45 00 01 0A 00 00 0A 00 01 02 00
FF FF FA 7D 1B 0A 00 0A 00 46 00 01 00 00 0B 02 05 00 00 FE 07 00 FC 01 00 DC 03 00 DC 02 00 DC
FF FF FA 7D 1B 02 00 02 00 47 00 01 00 00 8B 03 05 00 00 FE 07 00 FC 03 00 DC 04 00 DC 05 00 DC
FF FF FA 7D 1B 05 00 05 00 47 00 01 00 00 98 03 05 00 00 FE 07 00 FC 06 00 DC 08 00 DC 09 00 DC
FF FF FA 7D 1B 01 00 01 00 47 00 01 00 00 72 03 05 00 00 FE 07 00 FC 06 00 DC 08 00 DC 05 00 DC
FF FF FA 7D 1B 03 00 03 00 47 00 01 00 00 6A 03 05 07 00 FC 06 00 DC 08 00 DC 09 00 DC 01 00 DC
FF FF FA 7D 1B 04 00 04 00 47 00 01 00 00 64 03 05 07 00 FC 00 00 F4 03 00 DC 09 00 DC 05 00 DC
FF FF FA 7D 1B 06 00 06 00 47 00 01 00 00 95 03 05 00 00 FE 07 00 FC 09 00 DC 05 00 DC 01 00 DC
FF FF FA 7D 1B 07 00 07 00 46 00 01 00 00 6E 03 05 00 00 FC 09 00 DC 06 00 DC 03 00 DC 04 00 DC
FF FF FA 7D 1B 08 00 08 00 47 00 01 00 00 91 03 05 00 00 FE 07 00 FB 05 00 DC 09 00 DC 03 00 DC